

PROCESS IDENTIFICATION AND PID CONTROL

Su Whan Sung
Jietae Lee

Kyungpook National University, Republic of Korea

In-Beum Lee

Pohang University of Science and Technology, Republic of Korea



IEEE PRESS

IEEE Communications Society, Sponsor



John Wiley & Sons (Asia) Pte Ltd

PROCESS IDENTIFICATION AND PID CONTROL

PROCESS IDENTIFICATION AND PID CONTROL

Su Whan Sung
Jietae Lee

Kyungpook National University, Republic of Korea

In-Beum Lee

Pohang University of Science and Technology, Republic of Korea



IEEE PRESS

IEEE Communications Society, Sponsor



John Wiley & Sons (Asia) Pte Ltd

Copyright © 2009

John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop, # 02-01,
Singapore 129809

Visit our Home Page on www.wiley.com

MATLAB[®] and Simulink[®] are trademarks of The MathWorks, Inc. and are used with permission. The MathWorks does not warrant the accuracy of the text or exercises in this book. This book's use or discussion of MATLAB[®] and Simulink[®] software or related products does not constitute endorsement or sponsorship by The MathWorks of a particular pedagogical approach or particular use of the MATLAB[®] and Simulink[®] software.

All Rights Reserved. No part of this publication may be reproduced, stored in a retrieval system or transmitted in any form or by any means, electronic, mechanical, photocopying, recording, scanning, or otherwise, except as expressly permitted by law, without either the prior written permission of the Publisher, or authorization through payment of the appropriate photocopy fee to the Copyright Clearance Center. Requests for permission should be addressed to the Publisher, John Wiley & Sons (Asia) Pte Ltd, 2 Clementi Loop, #02-01, Singapore 129809, tel: 65-64632400, fax: 65-64646912, email: enquiry@wiley.com.

Designations used by companies to distinguish their products are often claimed as trademarks. All brand names and product names used in this book are trade names, service marks, trademarks or registered trademarks of their respective owners. The Publisher is not associated with any product or vendor mentioned in this book. All trademarks referred to in the text of this publication are the property of their respective owners.

This publication is designed to provide accurate and authoritative information in regard to the subject matter covered. It is sold on the understanding that the Publisher is not engaged in rendering professional services. If professional advice or other expert assistance is required, the services of a competent professional should be sought.

Other Wiley Editorial Offices

John Wiley & Sons, Ltd, The Atrium, Southern Gate, Chichester, West Sussex, PO19 8SQ, UK

John Wiley & Sons Inc., 111 River Street, Hoboken, NJ 07030, USA

Jossey-Bass, 989 Market Street, San Francisco, CA 94103-1741, USA

Wiley-VCH Verlag GmbH, Boschstrasse 12, D-69469 Weinheim, Germany

John Wiley & Sons Australia Ltd, 42 McDougall Street, Milton, Queensland 4064, Australia

John Wiley & Sons Canada Ltd, 5353 Dundas Street West, Suite 400, Toronto, ONT, M9B 6H8, Canada

Wiley also publishes its books in a variety of electronic formats. Some content that appears in print may not be available in electronic books.

Library of Congress Cataloging-in-Publication Data

Sung, Su Whan.

Process identification and PID control / Su Whan Sung, Jietae Lee, In-Beum Lee.
p. cm.

Includes bibliographical references and index.

ISBN 978-0-470-82410-8 (cloth)

1. Process control. 2. System identification. 3. PID controllers. I. Lee, Jietae. II. Lee, In, 1958- III. Title.

TS156.8.P7585 2009

629.8-dc22

2009001953

ISBN 978-0-470-82410-8 (HB)

Typeset in 10/12pt Times by Thomson Digital, Noida, India.

Printed and bound in Singapore by Markono Print Media Pte Ltd, Singapore.

This book is printed on acid-free paper responsibly manufactured from sustainable forestry in which at least two trees are planted for each one used for paper production.

To our wives and children

Contents

Preface	xi
Part One Basics of Process Dynamics	1
1 Mathematical Representations of Linear Processes	3
1.1 Introduction to Process Control and Identification	3
1.2 Properties of Linear Processes	9
1.3 Laplace Transform	16
1.4 Transfer Function and State-Space Systems	32
Problems	38
2 Simulations	45
2.1 Simulating Processes Composed of Differential Equations	45
2.2 Simulating Processes Including Time Delay	50
2.3 Simulating Closed-Loop Control Systems	57
2.4 Useful Numerical Analysis Methods	59
Problems	74
3 Dynamic Behavior of Linear Processes	79
3.1 Low-Order Plus Time-Delay Processes	79
3.2 Process Reaction Curve Method	84
3.3 Poles and Zeroes	86
3.4 Block Diagram	92
3.5 Frequency Responses	94
Problems	103
Part Two Process Control	109
4 Proportional–Integral–Derivative Control	111
4.1 Structure of Proportional–Integral–Derivative Controllers and Implementation in Computers/Microprocessors	111
4.2 Roles of Three Parts of Proportional–Integral–Derivative Controllers	122
4.3 Integral Windup	129

4.4 Commercial Proportional–Integral–Derivative Controllers	135
Problems	147
5 Proportional–Integral–Derivative Controller Tuning	151
5.1 Trial-and-Error Tuning	151
5.2 Simple Process Identification Methods	154
5.3 Ziegler–Nichols Tuning Rule	157
5.4 Internal Model Control Tuning Rule	159
5.5 Integral of the Time-Weighted Absolute Value of the Error Tunning Rule for a First-Order Plus Time-Delay Model (ITAE-1)	161
5.6 Integral of the Time-Weighted Absolute Value of the Error Tunning Rule for a Second-Order Plus Time-Delay Model (ITAE-2)	166
5.7 Optimal Gain Margin Tuning Rule for an Unstable Second-Order Plus Time-Delay Model (OGM-unstable)	169
5.8 Model Reduction Method for Proportional–Integral–Derivative Controller Tuning	170
5.9 Consideration of Modeling Errors	196
5.10 Concluding Remarks	196
Problems	197
6 Dynamic Behavior of Closed-Loop Control Systems	201
6.1 Closed-Loop Transfer Function and Characteristic Equation	201
6.2 Bode Stability Criterion	203
6.3 Nyquist Stability Criterion	207
6.4 Gain Margin and Phase Margin	210
Problems	212
7 Enhanced Control Strategies	215
7.1 Cascade Control	215
7.2 Time-Delay Compensators	217
7.3 Gain Scheduling	225
7.4 Proportional–Integral–Derivative Control using Internal Feedback Loop	228
Problems	231
Part Three Process Identification	233
8 Process Identification Methods for Frequency Response Models	235
8.1 Fourier Series	235
8.2 Frequency Response Analysis and Autotuning	240
8.3 Describing Function Analysis	241
8.4 Fourier Analysis	247
8.5 Modified Fourier Transform	250
8.6 Frequency Response Analysis with Integrals	261
Problems	271

9 Process Identification Methods for Continuous-Time Differential Equation Models	275
9.1 Identification Methods Using Integral Transforms	275
9.2 Prediction Error Identification Method	291
Problems	315
10 Process Identification Methods for Discrete-Time Difference Equation Models	317
10.1 Prediction Model: Autoregressive Exogenous Input Model and Output Error Model	317
10.2 Prediction Error Identification Method for the Autoregressive Exogenous Input Model	319
10.3 Prediction Error Identification Method for the Output Error Model	325
10.4 Concluding Remarks	335
Problems	336
11 Model Conversion from Discrete-Time to Continuous-Time Linear Models	337
11.1 Transfer Function of Discrete-Time Processes	337
11.2 Frequency Responses of Discrete-Time Processes and Model Conversion	338
Problems	342
Part Four Process Activation	343
12 Relay Feedback Methods	345
12.1 Conventional Relay Feedback Methods	345
12.2 Relay Feedback Method to Reject Static Disturbances	352
12.3 Relay Feedback Method under Nonlinearity and Static Disturbances	357
12.4 Relay Feedback Method for a Large Range of Operation	365
Problems	370
13 Modifications of Relay Feedback Methods	373
13.1 Process Activation Method Using Pulse Signals	373
13.2 Process Activation Method Using Sine Signals	387
Problems	397
Appendix Use of Virtual Control System	399
A.1 Setup of the Virtual Control System	399
A.2 Examples	400
Index	409

Preface

This book focuses on the basics of process control, process identification, PID controllers and autotuning. Our objective is to enable students and engineers who are not familiar with these topics to understand the basic concepts of feedback control, process identification, autotuning and design of real feedback controllers (especially PID controllers).

Parts One and Two are aimed at undergraduate students who have not taken any courses on process control. Parts Three and Four are appropriate for graduate students and control engineers who want to design real feedback controllers or perform research on process identification and autotuning. Parts One and Two introduce the basics of process control and dynamics, the analysis tools (Bode plot, Nyquist plot) to characterize the dynamics of the process, PID controllers and tuning, and advanced control strategies that have been widely used in industry. Also, simple simulation techniques required for practical controller designs and research on process identification and autotuning are also included. Part Three provides useful process identification methods actually used in industry. It includes several important identification algorithms to obtain frequency models or continuous-time/discrete-time transfer function models from the measured process input and output data sets. Part Four introduces various relay feedback methods to activate the process effectively for process identification and controller autotuning.

We have tried to include as many examples as possible. In particular, the readers can use the numerical examples and the MATLAB R codes with slight modifications to solve actual problems in their processes or research. The codes (MATLAB Rm-files) and real-time virtual processes for the simulations and practices are available from the Wiley website at www.wiley.com/go/swsung. The codes will be useful to those who want to understand the actual implementation techniques for control, process identification and autotuning. Also, the readers can design their own controllers, implement them and confirm the performances in real time using real-time virtual processes. Also, the problem-solving ability of students can be enhanced by performing a controller design project on the basis of the virtual process. We welcome the comments of students and instructors to improve the book and the materials for lectures and simulations. Please visit our other website at <http://pse.knu.ac.kr> for comments and questions about this book or process systems engineering. We hope this book is useful to you.

We wish to express special thanks to the students at KNU who provided the simulation results and detailed reviews: Cheol Ho Je, Chun Ho Jeon and Yu Jin Cheon. We acknowledge

John Wiley & Sons, especially James Murphy, Roger Bullen, Sarah Abdul Karim and Peter Lewis, for their effective cooperation and great care in preparing this book. We also gratefully acknowledge the financial support by Kyungpook National University (KNU Research Fund, 2006).

Su Whan Sung
Jietae Lee
In-Beum Lee

Part One

Basics of Process Dynamics

Part One introduces the basics of process dynamics which are appropriate for an undergraduate course. Chapter 1 defines linear processes and discusses how to represent linear processes in a mathematical way. Chapter 2 introduces several simulation and numerical analysis techniques required to simulate/design process controllers. Chapter 3 discusses the dynamic behaviors of linear processes and provides several analysis tools to characterize the dynamics of the control system.

1

Mathematical Representations of Linear Processes

1.1 Introduction to Process Control and Identification

The basic concepts and terms of process control and identification are first introduced.

1.1.1 Process Control

Process control consists of manipulating variables, controlled variables and processes. The manipulating variables and the controlled variables usually correspond to the process inputs and the process outputs respectively. The objective of process control is to make the process outputs (controlled variables) behave in a desired way by adjusting the process inputs (manipulating variables). Consider the temperature control system in Figure 1.1.

The SCR unit is to provide electrical power to the heating coil, which is proportional to the voltage $u(t)$. The temperature is measured by the thermocouple sensor. The objective of the temperature control system in Figure 1.1 is to drive the temperature $y(t)$ to the desired value by adjusting $u(t)$. So, $u(t)$ and $y(t)$ are the process input (manipulating variable) and the process output (controlled variable) respectively. The role of the feedback controller is to determine $u(t)$ appropriately on the basis of the measured $y(t)$ to achieve the control objective.

Example 1.1

Consider the control system in Figure 1.2. It consists of two tanks, a control valve, a DP cell and a controller. The DP cell and the control valve are to measure the liquid level of the last tank and adjust the inlet flow rate respectively. The objective of the control system is to drive the liquid level of the last tank to a desired value. In this case, the manipulating variable is the inlet flow rate and the controlled variable is the level of the last tank.

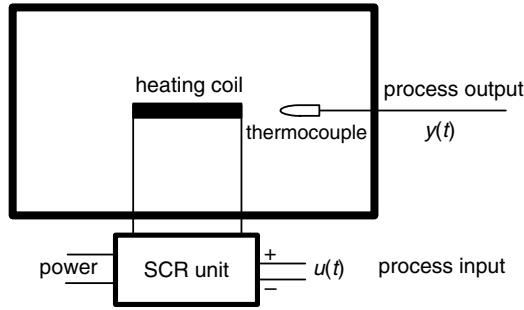


Figure 1.1 Temperature control system.

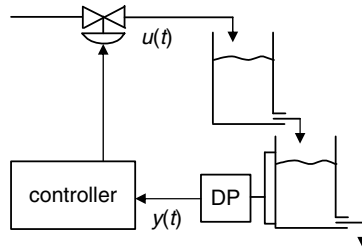


Figure 1.2 Level control system.

1.1.2 Process Identification

Process identification is the obtaining of a model of which the role is to predict the behavior of the process output for a given process input. The models are in the form of differential equations or frequency data sets (which will be explained later). From the energy balance equation for the temperature control system in Figure 1.1, the model of the following simple differential equation form can be derived:

$$\tau \frac{dy(t)}{dt} + y(t) = ku(t) + b \quad (1.1)$$

where τ , k and b are known constants determined by the heat capacity, mass, amplification coefficient, heat transfer coefficient, area and ambient temperature. This is a simple example of process identification. The behavior of $y(t)$ can be predicted by solving the differential equation for a given $u(t)$. In this book, how to obtain the model from historical data of the process input and the process output will be treated without considering physical principles such as material balance, energy balance and chemical reactions. This kind of model is called a “black-box model.”

Example 1.2

Assume that the black-box model structure for a given process has the following form:

$$\tau \frac{dy(t)}{dt} + y(t) = ku(t) \quad (1.2)$$

And assume that $y(t) = 1 - \exp(-2t)$ is obtained from an experiment when $u(t) = 1$ is applied to the process. Then, it is straightforward to estimate the model parameters of τ and k from the experiment. Replace $y(t)$ and $u(t)$ in (1.2) by $y(t) = 1 - \exp(-2t)$ and $u(t) = 1$. Then, (1.2) becomes $(2\tau - 1) \exp(-2t) + 1 = k$. So, $\tau = 0.5$ and $k = 1$ is obtained. This is a simple example of parameter estimation. The determination of the model structure and the parameter estimation are the core parts of process identification.

Example 1.3

Assume that the black-box model structure for a given process has the following form:

$$\tau^2 \frac{d^2y(t)}{dt^2} + 2\tau \frac{dy(t)}{dt} + y(t) = ku(t) \quad (1.3)$$

And assume that $y(t) = 0.5 \sin(t - \pi/2)$ is obtained from an experiment when $u(t) = \sin(t)$ is applied to the process. Estimate the model parameters τ and k from the experiment.

Solution Replace $y(t)$ and $u(t)$ in (1.3) by $y(t) = 0.5 \sin(t - \pi/2)$ and $u(t) = \sin(t)$. Then, (1.3) becomes

$$-0.5\tau^2 \sin(t - \pi/2) + \tau \cos(t - \pi/2) + 0.5 \sin(t - \pi/2) = k \sin(t)$$

which can be rewritten as $(0.5\tau^2 - 0.5) \cos(t) + \tau \sin(t) = k \sin(t)$ because $\sin(t - \pi/2) = -\cos(t)$ and $\cos(t - \pi/2) = \sin(t)$. So, $\tau = 1.0$ and $k = 1$ is obtained.

1.1.3 Steady State

When all the derivatives of the process input and process output are zero, this is called the steady state. For example, the process (1.4) will be (1.5) at steady state:

$$\frac{d^2y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = \frac{du(t)}{dt} + 2u(t) + 2 \quad (1.4)$$

$$\frac{d^2y_{ss}(t)}{dt^2} + 2 \frac{dy_{ss}(t)}{dt} + y_{ss}(t) = \frac{du_{ss}(t)}{dt} + 2u_{ss}(t) + 2 \rightarrow y_{ss}(t) = 2u_{ss}(t) + 2 \quad (1.5)$$

where the subscript 'ss' denotes steady state. As shown in (1.5), all the derivatives go to zeroes at steady state. On the other hand, a cyclic steady state means that the process output and input are periodic signals.

Example 1.4

Consider the process input $u(t)$ and the process output $y(t)$ in Figure 1.3. It can be seen that the process is in steady state after $t = 8$.

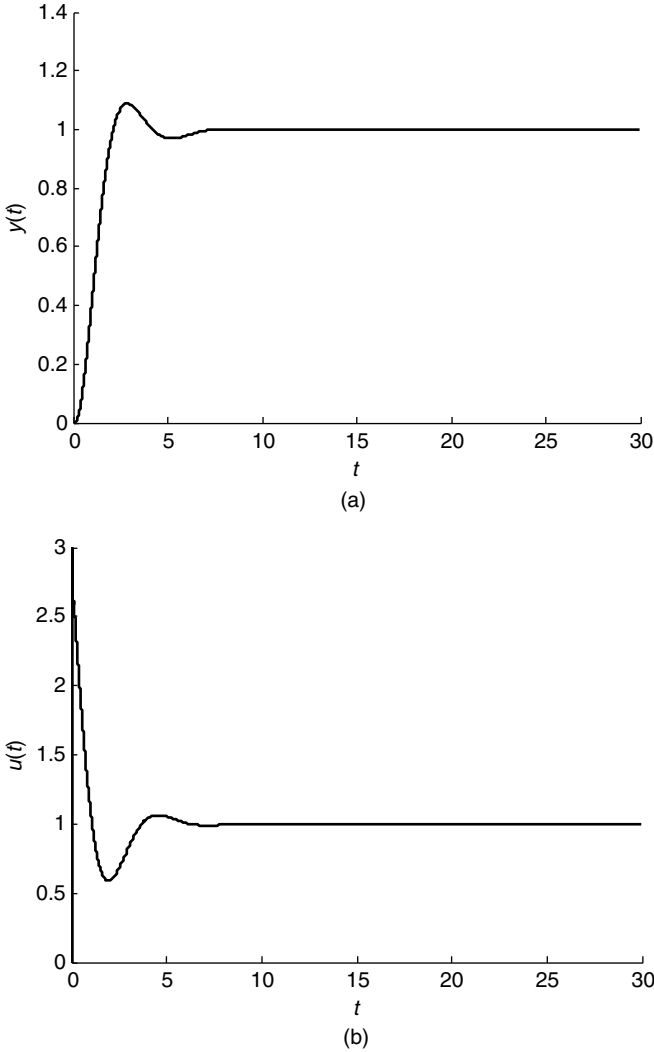


Figure 1.3 The process output and the process input of a control system.

Example 1.5

Obtain $y(t)$ for $u(t) = 2.0$ at steady state for the following process:

$$0.2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} (0.1 + 0.05 u(t)) + y(t) = \frac{du(t)}{dt} + \sqrt{u(t)} \quad (1.6)$$

Because all the derivatives are zero at steady state, $y_{ss}(t) = \sqrt{u_{ss}(t)}$. So, $y_{ss}(t) = \sqrt{2.0}$ for $u_{ss}(t) = 2.0$ at steady state.

Example 1.6

Obtain $y(t)$ for $y_s(t) = 1.0$ at steady state for the following process:

$$\frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = 0.1 \frac{du(t)}{dt} + u(t) \quad (1.7)$$

$$u(t) = 1.5(y_s(t) - y(t)) + 0.5 \frac{d(y_s(t) - y(t))}{dt} \quad (1.8)$$

Because all the derivatives in (1.7) are zero at steady state, $y_{ss}(t) = u_{ss}(t)$ and $u_{ss} = 1.5(y_{s,ss} - y_{ss})$ are obtained from (1.7) and (1.8). So, $y_{ss}(t) = 1.5/2.5$ at steady state.

Example 1.7

Consider the process input $u(t)$ and the process output $y(t)$ in Figure 1.4. It can be seen that the process is in cyclic steady state after about $t = 15$ because $u(t)$ and $y(t)$ are periodic after $t = 15$.

1.1.4 Deviation Variables

The deviation variable $\bar{x}(t)$ is the difference between the original variable $x(t)$ and a reference value x_{ref} . That is, $\bar{x}(t) = x(t) - x_{ref}$. So, it represents how far the original variable deviates from the reference value. The deviation variables for the process output and process input can be defined like $\bar{y}(t) = y(t) - y_{ref}$ and $\bar{u}(t) = u(t) - u_{ref}$ respectively. Here, y_{ref} and u_{ref} are usually the process output and the process input at steady state if there is no special notice. Note, y_{ref} is automatically fixed for the given u_{ref} at steady state. For example, the process (1.4) can be rewritten using deviation variables by subtracting (1.5) from (1.4):

$$\frac{d^2 \bar{y}(t)}{dt^2} + 2 \frac{d\bar{y}(t)}{dt} + \bar{y}(t) = \frac{d\bar{u}(t)}{dt} + 2\bar{u}(t) \quad (1.9)$$

$$\bar{y}(t) = y(t) - y_{ss}, \quad \bar{u}(t) = u(t) - u_{ss} \quad (1.10)$$

where $\bar{u}(t)$ and $\bar{y}(t)$ are deviation variables. u_{ss} and y_{ss} are the reference values for $u(t)$ and $y(t)$ respectively. Here, u_{ss} and y_{ss} should satisfy (1.5). So, y_{ss} is automatically fixed for the given u_{ss} at steady state.

Example 1.8

Rewrite the following process with deviation variables when the reference value for the process input $u(t)$ is chosen as 2.0.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) + 1 = 2 \frac{du(t)}{dt} + 3u(t) \quad (1.11)$$

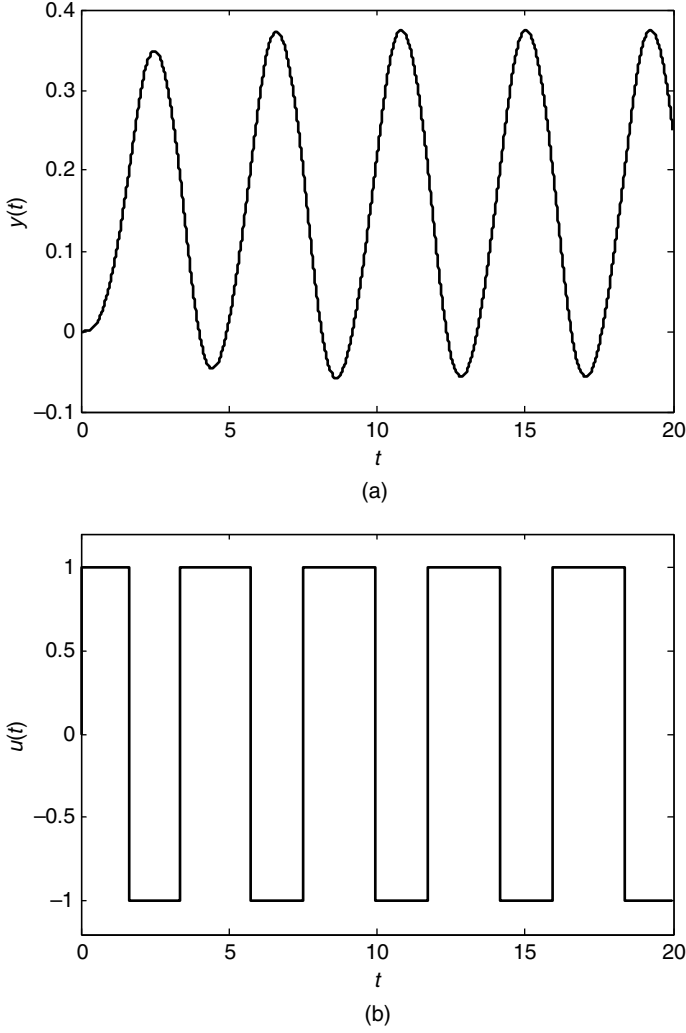


Figure 1.4 The process output and the process input of a relay feedback system.

Solution First, apply the steady-state assumption to (1.11):

$$\frac{d^3 y_{ss}(t)}{dt^3} + 3 \frac{d^2 y_{ss}(t)}{dt^2} + 3 \frac{dy_{ss}(t)}{dt} + y_{ss}(t) + 1 = 2 \frac{du_{ss}(t)}{dt} + 3u_{ss}(t) \quad (1.12)$$

By subtracting (1.12) from (1.11), the following process described by the deviation variables is obtained:

$$\frac{d^3 \bar{y}(t)}{dt^3} + 3 \frac{d^2 \bar{y}(t)}{dt^2} + 3 \frac{d\bar{y}(t)}{dt} + \bar{y}(t) = 2 \frac{d\bar{u}(t)}{dt} + 3\bar{u}(t) \quad (1.13)$$

$$\bar{y}(t) = y(t) - y_{ss}, \quad \bar{u}(t) = u(t) - u_{ss} \quad (1.14)$$

Here, $u_{ss} = 2.0$. From (1.11), it is known that $y_{ss} = 5.0$ for $u_{ss} = 2.0$ by applying the steady-state assumption. So, the deviation variables (1.14) should be

$$\bar{y}(t) = y(t) - 5.0, \quad \bar{u}(t) = u(t) - 2.0 \quad (1.15)$$

Example 1.9

Rewrite the following process with deviation variables when the reference value for the process input $u(t)$ is chosen as 2.0:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) + 1 = 2 \frac{du(t-0.5)}{dt} + 3u(t-0.5) \quad (1.16)$$

First, apply the steady-state assumption to (1.16):

$$\frac{d^3 y_{ss}(t)}{dt^3} + 3 \frac{d^2 y_{ss}(t)}{dt^2} + 3 \frac{dy_{ss}(t)}{dt} + y_{ss}(t) + 1 = 2 \frac{du_{ss}(t-0.5)}{dt} + 3u_{ss}(t-0.5) \quad (1.17)$$

By subtracting (1.17) from (1.16) the following process described by the deviation variables is obtained:

$$\frac{d^3 \bar{y}(t)}{dt^3} + 3 \frac{d^2 \bar{y}(t)}{dt^2} + 3 \frac{d\bar{y}(t)}{dt} + \bar{y}(t) = 2 \frac{d\bar{u}(t-0.5)}{dt} + 3\bar{u}(t-0.5) \quad (1.18)$$

$$\bar{y}(t) = y(t) - y_{ss}, \quad \bar{u}(t) = u(t) - u_{ss} \quad (1.19)$$

From (1.16) $y_{ss} = 5.0$ is obtained for $u_{ss} = 2.0$ because $u_{ss}(t) = u_{ss}(t-0.5) = 2.0$ at steady state. So, the deviation variables (1.19) should be

$$\bar{y}(t) = y(t) - 5.0, \quad \bar{u}(t) = u(t) - 2.0 \quad (1.20)$$

1.2 Properties of Linear Processes

Linear processes are defined and several important properties of linear processes are discussed.

1.2.1 Linear Process

When the dynamics of a process can be described by a linear combination of derivatives ($d^j y(t)/dt^j$, $d^j u(t)/dt^j$, $j = 0, 1, 2, \dots$) of the process output $y(t)$ and the process input $u(t)$ and a constant, it is a linear process. If the coefficients are time invariant (constants), then it is the time-invariant linear process. If the coefficients are time variant, then it is the time-variant linear process. For example, (1.4) is a linear process. But, the following processes are nonlinear:

$$3 \frac{d^2 y(t)}{dt^2} - \frac{dy(t)}{dt} + y(t) = \frac{du(t)}{dt} u(t) + 4u(t) \quad (1.21)$$

$$\frac{dy(t)}{dt} + y(t) = 4\sqrt{u(t)} \quad (1.22)$$

Equations (1.21) and (1.22) are nonlinear because of the $(du(t)/dt)u(t)$ and $\sqrt{u(t)}$ terms respectively.

Example 1.10

Consider the following process:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (1.23)$$

Here, it should be noted that

$$u(t - 0.5) = u(t) + \sum_{i=1}^{\infty} \frac{(-0.5)^i}{i!} \frac{d^i u(t)}{dt^i}$$

(which will be discussed later). So, (1.23) is a time-invariant linear process. That is, linear processes can include time delays.

Example 1.11

Consider the following process:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (1.24)$$

$$u(t) = 0.5 \int_0^t (1 - y(\tau)) d\tau + 0.1 \frac{d(1 - y(t))}{dt} \quad (1.25)$$

In Example 1.11, it is revealed that the time delay does not change the linearity. Also, by differentiating (1.24) and (1.25), the integral in (1.25) then disappears. So, (1.24) and (1.25) is a time-invariant linear process.

Example 1.12

Consider the process

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (1.26)$$

$$u(t) = 2(y_s(t) - y(t)) \quad (1.27)$$

From (1.26) and (1.27), the following process is obtained:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 2[y_s(t - 0.5) - y(t - 0.5)] \quad (1.28)$$

So, the process (1.28) of which the input and output are $y_s(t)$ and $y(t)$ is a time-invariant linear process.

Example 1.13

Consider the process

$$\frac{d^2y(t)}{dt^2} + (2 + 0.1t) \frac{dy(t)}{dt} + (1 - 0.05t)y(t) = (2 + 0.3t)u(t) \quad (1.29)$$

in which the coefficients are time variant. Thus, this is a time-variant linear process.

1.2.2 Superposition Rule

Suppose that the process input is a linear combination of several signals. Then, the process output is the linear combination of the respective process outputs for the several signals if the process is linear. For example, the process output $y(t)$ for the process input $u(t) = u_1(t) + 0.3u_2(t) + 1.3u_3(t)$ can be obtained without a plant test from the available information that the process is linear and the process outputs $y_1(t)$, $y_2(t)$ and $y_3(t)$ are the responses of the process to the process inputs $u_1(t)$, $u_2(t)$ and $u_3(t)$ respectively. That is, it is clear that the process output is $y(t) = y_1(t) + 0.3y_2(t) + 1.3y_3(t)$ for the given process input $u(t)$ by the superposition rule (Figure 1.5).

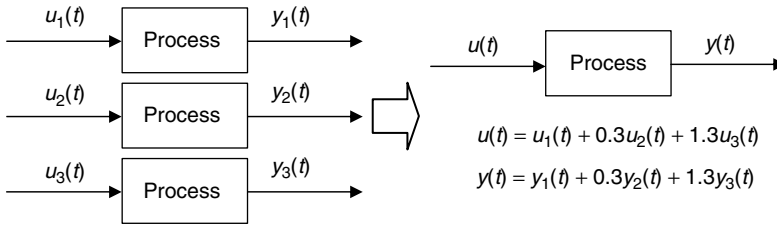


Figure 1.5 Superposition principle.

Therefore, if the pairs $(u_1(t), y_1(t))$, $(u_2(t), y_2(t))$, $(u_3(t), y_3(t))$, ... for the given linear process are known, then $y(t)$ can be easily calculated corresponding to any $u(t)$ of a linear combination $u_1(t)$, $u_2(t)$, $u_3(t)$, ...

Example 1.14

Obtain the process output $y(0.0)$, $y(0.1)$, $y(0.2)$, $y(0.3)$ of a linear process for the following process input $u(t)$:

$$u(t) = 2 \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (1.30)$$

The available information is that the responses of the process to the process input $u_1(t) = 1$ for $t \geq 0$, $u_1(t) = 0$ for $t < 0$ are $y_1(0.0) = 0.0$, $y_1(0.1) = 0.01$, $y_1(0.2) = 0.02$ and $y_1(0.3) = 0.04$.

Solution Note that $u(t) = 2u_1(t)$. Then, $y(t) = 2y_1(t)$ by the superposition rule. So, $y(0.0) = 0.0$, $y(0.1) = 0.02$, $y(0.2) = 0.04$ and $y(0.3) = 0.08$ are obtained.

Example 1.15

Obtain the process output $y(0.0)$, $y(0.1)$, $y(0.2)$, $y(0.3)$ of a linear process for the following process input $u(t)$:

$$u(t) = 1 \quad \text{for } t \geq 0.1, \quad u(t) = 2 \quad \text{for } 0 \leq t < 0.1, \quad u(t) = 0 \quad \text{for } t < 0 \quad (1.31)$$

The available information is that the responses of the process to the process input $u_1(t) = 1$ for $t \geq 0$, $u_1(t) = 0$ for $t < 0$ are $y_1(0.0) = 0.0$, $y_1(0.1) = 0.01$, $y_1(0.2) = 0.02$ and $y_1(0.3) = 0.04$, and the responses for the process input $u_2(t) = 1$ for $t \geq 0.1$, $u_2(t) = 0$ for $t < 0.1$ are $y_2(0.0) = 0.0$, $y_2(0.1) = 0.0$, $y_2(0.2) = 0.01$ and $y_2(0.3) = 0.02$.

Solution Note that $u(t) = 2u_1(t) - u_2(t)$. Then, $y(t) = 2y_1(t) - y_2(t)$ by the superposition rule. So, $y(0.0) = 0.0$, $y(0.1) = 0.02$, $y(0.2) = 0.03$ and $y(0.3) = 0.06$ are obtained.

Example 1.16

Obtain the process output $y(0.0)$, $y(0.1)$, $y(0.2)$, $y(0.3)$ of a linear time-invariant process for the following process input $u(t)$:

$$u(t) = 0 \quad \text{for } t \geq 0.1, \quad u(t) = 1 \quad \text{for } 0 \leq t < 0.1, \quad u(t) = 0 \quad \text{for } t < 0 \quad (1.32)$$

The available information is that the responses of the process to the process input $u_1(t) = 1$ for $t \geq 0$, $u_1(t) = 0$ for $t < 0$ are $y_1(-0.1) = 0.0$, $y_1(0.0) = 0.0$, $y_1(0.1) = 0.01$, $y_1(0.2) = 0.02$ and $y_1(0.3) = 0.04$.

Solution Note that $u(t) = u_1(t) - u_1(t - 0.1)$. Then, $y(t) = y_1(t) - y_1(t - 0.1)$ by the superposition rule. So, $y(0.0) = 0.0$, $y(0.1) = 0.01$, $y(0.2) = 0.01$ and $y(0.3) = 0.02$ are obtained. This example demonstrates how to obtain the impulse responses from the step responses.

Example 1.17

Obtain the process output $y(0.0)$, $y(0.1)$, $y(0.2)$, $y(0.3)$ of a linear time-invariant process for the following process input $u(t)$:

$$u(t) = 3 \quad \text{for } 0.2 \leq t, \quad u(t) = 4 \quad \text{for } 0.1 \leq t < 0.2, \quad u(t) = 2 \quad \text{for } 0 \leq t < 0.1, \quad u(t) = 0 \quad \text{for } t < 0 \quad (1.33)$$

The available information is that the responses of the process to the process input $u_1(t) = 0$ for $t \geq 0.1$, $u_1(t) = 1$ for $0 \leq t < 0.1$, $u_1(t) = 0$ for $t < 0$ are $y_1(-0.2) = 0.0$, $y_1(-0.1) = 0.0$, $y_1(0.0) = 0.0$, $y_1(0.1) = 0.01$, $y_1(0.2) = 0.03$ and $y_1(0.3) = 0.02$.

Solution Note that $u(t) = 3u_1(t - 0.2) + 4u_1(t - 0.1) + 2u_1(t)$. Then, $y(t) = 3y_1(t - 0.2) + 4y_1(t - 0.1) + 2y_1(t)$ by the superposition rule. So, $y(0.0) = 0.0$, $y(0.1) = 0.02$, $y(0.2) = 0.10$ and $y(0.3) = 0.19$ are obtained. This example demonstrates how to calculate the process output from the impulse responses of the process. This kind of model is called an “impulse response model.”

Example 1.18

Obtain the process output $y(t)$ of a linear process for the process input $u(t) = 3 \sin(t) + 2 \sin(3t)$. The available information is that the responses of the process to the process input $u_1(t) = \sin(t)$ are $y_1(t) = 0.3 \sin(t - 0.1)$ and the responses of the process for the process input $u_2(t) = \sin(3t)$ are $y_2(t) = 0.1 \sin(3t - 0.2)$.

Solution Note that $u(t) = 3u_1(t) + 2u_2(t)$. Then, $y(t) = 3y_1(t) + 2y_2(t)$ by the superposition rule. So, $y(t) = 0.9 \sin(t - 0.1) + 0.2 \sin(3t - 0.2)$ is obtained.

Example 1.19

Obtain the process output $y(t)$ of a linear time-invariant process for the process input $u(t) = \sin(t)$. The available information is that the responses of the process to the process input $u_1(t) = 0.4 \sin(t - 0.1) + 0.2 \sin(3t - 0.2)$ are $y_1(t) = 0.3 \sin(t - 0.2) + 0.1 \sin(3t - 0.4)$.

Solution $y(t) = 0.3 \sin(t - 0.2)$ is obtained for $u(t) = 0.4 \sin(t - 0.1)$ and, equivalently, $y(t) = 3 \sin(t - 0.1)/4$ for $u(t) = \sin(t)$ from the given information and the superposition rule. Also, $y(t) = \sin(3t - 0.2)/2$ is the response to the process input $u(t) = \sin(3t)$.

1.2.3 Linearization

It is notable that many nonlinear processes can be approximated effectively by linearized models. Linearization is the process of obtaining a linear model to approximate the nonlinear model. Taylor series are frequently used for linearization. Theoretically, a nonlinear function $f(u)$ can be represented by the following Taylor series:

$$f(u) = f(u_0) + \left. \frac{df}{du} \right|_{u=u_0} (u - u_0) + \frac{1}{2!} \left. \frac{d^2f}{du^2} \right|_{u=u_0} (u - u_0)^2 + \frac{1}{3!} \left. \frac{d^3f}{du^3} \right|_{u=u_0} (u - u_0)^3 + \dots \quad (1.34)$$

The following approximation of (1.34) to (1.35) is called linearization at $u = u_0$:

$$f(u) \approx f(u_0) + \left. \frac{df}{du} \right|_{u=u_0} (u - u_0) \quad (1.35)$$

For example, the straight line in Figure 1.6 corresponds to (1.35), which is close to (1.34) around $u = u_0$.

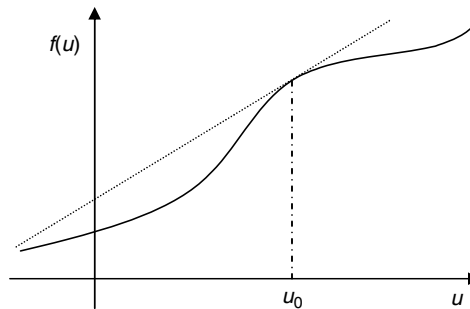


Figure 1.6 Linearization of $f(u)$ at $u = u_0$.

Equation (1.22) can be approximated by the Taylor series at $u(t) = u_0 = 1$ as follows:

$$\frac{dy(t)}{dt} + y(t) = 4\sqrt{u(t)} \approx 4\sqrt{u_0} + 4\frac{1}{2}(u_0)^{-1/2}(u(t) - u_0) \quad (1.36)$$

Equation (1.36) can be rewritten to the following linearized process:

$$\frac{dy(t)}{dt} + y(t) \approx 2\sqrt{u_0} + 2(u_0)^{-1/2}u(t) \quad (1.37)$$

Equation (1.37) can be also described by the deviation variables:

$$\frac{d\bar{y}(t)}{dt} + \bar{y}(t) = 2(u_0)^{-1/2}\bar{u}(t), \quad \bar{y}(t) = y(t) - y_{ss}, \quad \bar{u}(t) = u(t) - u_{ss} \quad (1.38)$$

Now, the linearized process (1.38) is obtained for the nonlinear process (1.22).

The Taylor series approximation can be also applied to multivariable nonlinear functions such as $f(u_1, u_2)$ as follows:

$$f(u_1, u_2) \approx f(u_{1,0}, u_{2,0}) + \left. \frac{\partial f}{\partial u_1} \right|_{u_1=u_{1,0}, u_2=u_{2,0}} (u_1 - u_{1,0}) + \left. \frac{\partial f}{\partial u_2} \right|_{u_1=u_{1,0}, u_2=u_{2,0}} (u_2 - u_{2,0}) \quad (1.39)$$

Similarly, the Taylor series approximation can be applied to multivariable functions of which the number of the variables is bigger than 2 in a straightforward manner.

Example 1.20

Obtain the linearized process around $u(t) = u_0 = 2$ for the following nonlinear process, and express it with the deviation variables:

$$\frac{dy(t)}{dt} + y^{1.5}(t) = u^3(t) \quad (1.40)$$

Solution Equation (1.40) becomes $y^{1.5}(t) = u^3(t)$ at steady state. So, the value of the process output $y(t)$ for $u(t) = u_0 = 2$ is $y_0 = 2^{3/1.5}$. We obtain $y^{1.5}(t) \approx 8 + 3(y(t) - 2^{3/1.5})$ and $u^3(t) \approx 8 + 12(u(t) - 2)$ by the Taylor series approximation. Then, the linearized process is

$$\frac{dy(t)}{dt} + 8 + 3(y(t) - 2^{3/1.5}) = 8 + 12(u(t) - 2) \quad (1.41)$$

Equation (1.41) is valid for the steady state. That is, the following equation is valid:

$$\frac{dy_0(t)}{dt} + 8 + 3(y_0(t) - 2^{3/1.5}) = 8 + 12(u_0(t) - 2) \quad (1.42)$$

So, the following linearized process represented by the deviation variables is obtained by subtracting (1.42) from (1.41):

$$\frac{d\bar{y}(t)}{dt} + 3\bar{y}(t) = 12\bar{u}(t) \quad (1.43)$$

$$\bar{y}(t) = y(t) - 2^{3/1.5}, \quad \bar{u}(t) = u(t) - 2 \quad (1.44)$$

Example 1.21

Obtain the linearized process around $u(t) = u_0 = 2$ for the following nonlinear process, and express it with the deviation variables:

$$\frac{dy(t)}{dt} + y(t)(1 + 0.1u^2(t)) = u^3(t) \quad (1.45)$$

Solution Equation (1.45) becomes $y(t)(1 + 0.1u^2(t)) = u^3(t)$ at steady state. So, the value of the process output $y(t)$ for $u(t) = u_0 = 2$ is $y_0 = 8/1.4$. The following equation is obtained by the Taylor series approximation for the multivariable function $y(t)(1 + 0.1u^2(t))$:

$$\begin{aligned} y(t)(1 + 0.1u^2(t)) &\approx y_0(1 + 0.1u_0^2) + (1 + 0.1u_0^2)(y(t) - y_0) + 0.2y_0u_0(u(t) - u_0) \\ &= 8 + 1.4\left(y(t) - \frac{8}{1.4}\right) + \frac{3.2}{1.4}(u(t) - 2) \end{aligned} \quad (1.46)$$

and $u^3(t) \approx 8 + 12(u(t) - 2)$ by the Taylor series approximation. Then, the linearized process is as follows:

$$\frac{dy(t)}{dt} + 8 + 1.4\left(y(t) - \frac{8}{1.4}\right) + \frac{3.2}{1.4}(u(t) - 2) = 8 + 12(u(t) - 2) \quad (1.47)$$

Equation (1.46) is valid for the steady state. That is, the following equation is valid:

$$\frac{dy_0(t)}{dt} + 8 + 1.4\left(y_0(t) - \frac{8}{1.4}\right) + \frac{3.2}{1.4}(u_0(t) - 2) = 8 + 12(u_0(t) - 2) \quad (1.48)$$

So, the following linearized process represented by the deviation variables is obtained by subtracting (1.48) from (1.47):

$$\frac{d\bar{y}(t)}{dt} + 1.4\bar{y}(t) = \left(12 - \frac{3.2}{1.4}\right)\bar{u}(t) \quad (1.49)$$

$$\bar{y}(t) = y(t) - 8/1.4, \quad \bar{u}(t) = u(t) - 2 \quad (1.50)$$

Example 1.22

Obtain the linearized process around $u(t) = u_0 = 2$ for the following nonlinear process and express it with the deviation variables:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) + 0.01\frac{dy(t)}{dt}u(t)y(t) = u(t) \quad (1.51)$$

Solution Equation (1.51) becomes $y(t) = u(t)$ at steady state. So, the value of the process output $y(t)$ for $u(t) = u_0 = 2$ is $y_0 = 2$ and the value of $dy(t)/dt$ at steady state is zero. So, the linearization should be done around $u_0 = 2$, $y_0 = 2$ and $(dy(t)/dt)_0 = 0$. The following equation is obtained by the Taylor series approximation for the multivariable function

$0.01(dy(t)/dt)u(t)y(t)$. Here, $dy(t)/dt$ should be considered one of the variables of the nonlinear function $0.01(dy(t)/dt)u(t)y(t)$. Also, $\partial(0.01(dy(t)/dt)u(t)y(t))/\partial u(t) = 0$ and $\partial(0.01(dy(t)/dt)u(t)y(t))/\partial y(t) = 0$ at steady state should be used.

$$0.01 \frac{dy(t)}{dt} u(t)y(t) \approx 0.01u_0y_0 \left(\frac{dy(t)}{dt} - 0 \right) = 0.04 \frac{dy(t)}{dt} \quad (1.52)$$

Then, the linearized process is as follows:

$$\frac{d^2y(t)}{dt^2} + 2.04 \frac{dy(t)}{dt} + y(t) = u(t) \quad (1.53)$$

Equation (1.53) is valid for the steady state. That is, the following equation is valid:

$$\frac{d^2y_0(t)}{dt^2} + 2.04 \frac{dy_0(t)}{dt} + y_0(t) = u_0(t) \quad (1.54)$$

So, the following linearized process represented by the deviation variables is obtained by subtracting (1.54) from (1.53):

$$\frac{d^2\bar{y}(t)}{dt^2} + 2.04 \frac{d\bar{y}(t)}{dt} + \bar{y}(t) = \bar{u}(t) \quad (1.55)$$

$$\bar{y}(t) = y(t) - 2, \quad \bar{u}(t) = u(t) - 2 \quad (1.56)$$

1.3 Laplace Transform

The Laplace transform plays an important role in analyzing/designing the control system. In this section, the definition of the Laplace transform is introduced. Also how to obtain the Laplace transforms for various functions and how to solve differential equations using the Laplace transform are explained.

1.3.1 Laplace Transforms, Inverse Laplace Transforms

The Laplace transform of $f(t)$ is defined as

$$L\{f(t)\} = f(s) = \int_0^{\infty} \exp(-st)f(t) dt \quad (1.57)$$

where s is a complex variable. $f(s)$ or $L\{f(t)\}$ denotes the Laplace transform of $f(t)$. Note that $f(s)$ is a function of s because it is the integral of $\exp(-st)f(t)$ from $t = 0$ to $t = \infty$, which means that the variable t disappears.

The inverse Laplace transform restores the original function $f(t)$ from the Laplace transform of $f(t)$:

$$L^{-1}\{f(s)\} = f(t) \quad (1.58)$$

The following examples demonstrate how to obtain the Laplace transforms for several functions. Also, several important properties of the Laplace transform are shown.

Example 1.23

$f(t) = 1$:

$$L\{f(t)\} = f(s) = \int_0^{\infty} \exp(-st) \, dt = - \left. \frac{\exp(-st)}{s} \right|_0^{\infty} = \frac{1}{s} \quad (1.59)$$

Example 1.24

$f(t) = e^{at}$:

$$L\{f(t)\} = f(s) = \int_0^{\infty} \exp(-st) \exp(at) \, dt = \int_0^{\infty} \exp[-(s-a)t] \, dt = - \left. \frac{\exp[-(s-a)t]}{s-a} \right|_0^{\infty} = \frac{1}{s-a} \quad (1.60)$$

Example 1.25

The Laplace transform satisfies the following linearity:

$$\begin{aligned} L\{ag(t) + bh(t)\} &= \int_0^{\infty} \exp(-st)(ag(t) + bh(t)) \, dt \\ &= a \int_0^{\infty} \exp(-st)g(t) \, dt + b \int_0^{\infty} \exp(-st)h(t) \, dt \\ &= aL\{g(t)\} + bL\{h(t)\} = ag(s) + bh(s) \end{aligned} \quad (1.61)$$

The linearity of the Laplace transform is used in Examples 1.27–1.29.

Example 1.26

$f(t) = \cosh(at) = \frac{\exp(at) + \exp(-at)}{2}$:

$$f(s) = \frac{1}{2}L\{\exp(at)\} + \frac{1}{2}L\{\exp(-at)\} = \frac{1}{2} \left(\frac{1}{s-a} + \frac{1}{s+a} \right) = \frac{s}{s^2 - a^2} \quad (1.62)$$

Example 1.27

$f(t) = \sinh(at) = \frac{\exp(at) - \exp(-at)}{2}$:

$$f(s) = \frac{1}{2}L\{\exp(at)\} - \frac{1}{2}L\{\exp(-at)\} = \frac{1}{2} \left(\frac{1}{s-a} - \frac{1}{s+a} \right) = \frac{a}{s^2 - a^2} \quad (1.63)$$

Example 1.28

$$\begin{aligned}
 f(t) = \cos(\omega t) &= \frac{\exp(i\omega t) + \exp(-i\omega t)}{2} : \\
 f(s) &= \frac{1}{2}L\{\exp(i\omega t)\} + \frac{1}{2}L\{\exp(-i\omega t)\} = \frac{1}{2}\left(\frac{1}{s-i\omega} + \frac{1}{s+i\omega}\right) \\
 &= \frac{1}{2}\left(\frac{s+i\omega}{s^2+\omega^2} + \frac{s-i\omega}{s^2+\omega^2}\right) = \frac{s}{s^2+\omega^2}
 \end{aligned} \tag{1.64}$$

Example 1.29

$$\begin{aligned}
 f(t) = \sin(\omega t) &= \frac{\exp(i\omega t) - \exp(-i\omega t)}{2i} : \\
 f(s) &= \frac{1}{2i}L\{\exp(i\omega t)\} - \frac{1}{2i}L\{\exp(-i\omega t)\} = \frac{1}{2i}\left(\frac{1}{s-i\omega} - \frac{1}{s+i\omega}\right) \\
 &= \frac{1}{2i}\left(\frac{s+i\omega}{s^2+\omega^2} - \frac{s-i\omega}{s^2+\omega^2}\right) = \frac{\omega}{s^2+\omega^2}
 \end{aligned} \tag{1.65}$$

Example 1.30

Relationship between $L\{\exp(at)f(t)\}$ and $L\{f(t)\}$:

$$L\{\exp(at)f(t)\} = \int_0^\infty \exp(-st)\exp(at)f(t) dt = \int_0^\infty \exp[-(s-a)t]f(t) dt = f(s-a) \tag{1.66}$$

The property of the Laplace transform of Example 1.30 is used in Example 1.31.

Example 1.31

$f(t) = \exp(at)\cos(\omega t)$ and $f(t) = \exp(at)\sin(\omega t)$:

$$L\{\exp(at)\cos(\omega t)\} = \frac{(s-a)}{(s-a)^2 + \omega^2}, \quad L\{\exp(at)\sin(\omega t)\} = \frac{\omega}{(s-a)^2 + \omega^2} \tag{1.67}$$

Example 1.32

Relationship between $L\{t^n\}$ and $L\{t^{n-1}\}$:

$$L\{t^n\} = \int_0^\infty t^n \exp(-st) dt = -t^n \frac{\exp(-st)}{s} \Big|_0^\infty + \frac{n}{s} \int_0^\infty t^{n-1} \exp(-st) dt = \frac{n}{s} L\{t^{n-1}\} \tag{1.68}$$

It is straightforward to obtain (1.69) from (1.68):

$$L\{t^n\} = \frac{n!}{s^{n+1}} \quad \text{if } n \text{ is an integer} \tag{1.69}$$

Example 1.33

Numerical estimation of $G(s) = y(s)/u(s)$ at $s = 3i$ for the given $u(t)$ and $y(t)$.

From the definition of the Laplace transform, we have (1.70) and (1.71):

$$u(s = 3i) = \int_0^{\infty} \exp(-3it)u(t) dt \quad (1.70)$$

$$y(s = 3i) = \int_0^{\infty} \exp(-3it)y(t) dt \quad (1.71)$$

Equations (1.70) and (1.71) can be numerically calculated by a numerical integration method if $u(t)$ and $y(t)$ converge to zero as t increases. Then, it is straightforward to calculate $G(3i) = y(3i)/u(3i)$. This example shows how to estimate the frequency responses of the process from the measured process input $u(t)$ and the process output $y(t)$. Detailed descriptions on numerical integration methods and frequency response estimation methods will be given later in this book.

1.3.2 Laplace Transforms for Derivatives and Integrals

The Laplace transform of the derivatives of a function can be expressed by the Laplace transform of the function and the initial conditions, as shown below.

$$\begin{aligned} L\left\{\frac{df(t)}{dt}\right\} &= \int_0^{\infty} \exp(-st) \frac{df(t)}{dt} dt = \exp(-st)f(t)\Big|_0^{\infty} + s \int_0^{\infty} \exp(-st)f(t) dt \\ &= sL\{f(t)\} - f(0) \end{aligned} \quad (1.72)$$

$$L\left\{\frac{df(t)}{dt}\right\} = sL\{f(t)\} - f(0) \quad (1.73)$$

$$L\left\{\frac{d^2f(t)}{dt^2}\right\} = sL\left\{\frac{df(t)}{dt}\right\} - \frac{df(t)}{dt}\Big|_{t=0} = s^2L\{f(t)\} - sf(0) - \frac{df(t)}{dt}\Big|_{t=0} \quad (1.74)$$

$$\begin{aligned} L\left\{\frac{d^3f(t)}{dt^3}\right\} &= sL\left\{\frac{d^2f(t)}{dt^2}\right\} - \frac{d^2f(t)}{dt^2}\Big|_{t=0} = s^3L\{f(t)\} - s^2f(0) - s\frac{df(t)}{dt}\Big|_{t=0} - \frac{d^2f(t)}{dt^2}\Big|_{t=0} \\ & \quad (1.75) \end{aligned}$$

$$L\left\{\frac{d^nf(t)}{dt^n}\right\} = s^nL\{f(t)\} - s^{n-1}f(0) - s^{n-2}\frac{df(t)}{dt}\Big|_{t=0} - \dots - \frac{d^{n-1}f(t)}{dt^{n-1}}\Big|_{t=0} \quad (1.76)$$

The Laplace transform of the integrals of a function can be expressed by the Laplace transform of the function, as shown in (1.77):

$$\begin{aligned} L\left\{\int_0^t f(\tau) d\tau\right\} &= \int_0^\infty \exp(-st) \int_0^t f(\tau) d\tau dt \\ &= -\frac{1}{s} \exp(-st) \int_0^t f(\tau) d\tau \Big|_0^\infty + \frac{1}{s} \int_0^\infty \exp(-st) f(t) dt = \frac{f(s)}{s} \end{aligned} \quad (1.77)$$

The following examples show how (1.76) is used to convert the differential equation in the time domain to the algebraic equation in the s domain.

Example 1.34

Consider the following differential equation:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = \frac{d^2 u(t)}{dt^2} + 3 \frac{du(t)}{dt} + u(t) \quad (1.78)$$

where the initial conditions are

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0$$

If the Laplace transform is applied to (1.78), then (1.79) is obtained by (1.76):

$$s^3 y(s) + 3s^2 y(s) + 3sy(s) + y(s) = s^2 u(s) + 3su(s) + u(s) \quad (1.79)$$

$$\frac{y(s)}{u(s)} = \frac{s^2 + 3s + 1}{s^3 + 3s^2 + 3s + 1} \quad (1.80)$$

Example 1.35

Consider the differential equation:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = \frac{d^2 u(t)}{dt^2} + 3 \frac{du(t)}{dt} + u(t) \quad (1.81)$$

where the initial conditions are

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = 0, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = 0.1, \quad y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = 1, \quad u(0) = 0$$

If the Laplace transform is applied to (1.81), then (1.82) is obtained by (1.76):

$$s^3 y(s) - 0.1s + 3s^2 y(s) - 0.3 + 3sy(s) + y(s) = s^2 u(s) - 1 + 3su(s) + u(s) \quad (1.82)$$

Example 1.36

Consider the following integro-differential equation:

$$\frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} + y(t) = -0.1 \frac{du(t)}{dt} + u(t) + \int_0^t u(\tau) d\tau \quad (1.83)$$

where the initial conditions are

$$\left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, u(0) = 0$$

If the Laplace transform is applied to (1.83), then (1.84) is obtained by (1.76) and (1.77):

$$s^2 y(s) + sy(s) + y(s) = -0.1su(s) + u(s) + \frac{u(s)}{s} \quad (1.84)$$

Example 1.37

The differential equation corresponding to the algebraic equation (1.85) is (1.86):

$$s^2 y(s) + 2sy(s) + y(s) = -su(s) + u(s) + 0.5 \frac{u(s)}{s} \quad (1.85)$$

$$\begin{aligned} \frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) &= -\frac{du(t)}{dt} + u(t) \\ + 0.5 \int_0^t u(\tau) d\tau, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} &= y(0) = 0, \quad u(0) = 0 \end{aligned} \quad (1.86)$$

1.3.3 Laplace Transform for Unit Step Function, Time Delay and Impulse Function

The definition of the unit step (Figure 1.7) function is

$$S(t - \theta) \equiv \begin{cases} 0 & \text{if } t \leq \theta \\ 1 & \text{if } t > \theta \end{cases} \quad (1.87)$$

The Laplace transform of the unit step function $S(t - \theta)$ is

$$L\{S(t - \theta)\} = \int_0^\infty \exp(-st) S(t - \theta) dt = \int_\theta^\infty \exp(-st) dt = -\left. \frac{\exp(-st)}{s} \right|_\theta^\infty = \frac{\exp(-\theta s)}{s} \quad (1.88)$$

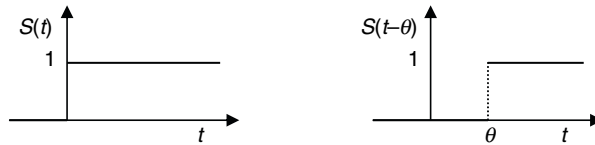


Figure 1.7 Unit step function.

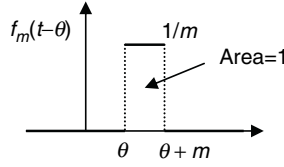


Figure 1.8 Impulse function.

The time-delayed function $f(t - \theta)S(t - \theta)$ has the following Laplace transform:

$$L\{f(t - \theta)S(t - \theta)\} = \int_0^{\infty} \exp(-st)f(t - \theta)S(t - \theta) dt = \int_{t=\theta}^{t=\infty} \exp(-st)f(t - \theta) dt \quad (1.89)$$

Substituting $x = t - \theta$, the following Laplace transform is obtained:

$$\begin{aligned} L\{f(t - \theta)S(t - \theta)\} &= \int_{x=0}^{x=\infty - \theta} \exp[-s(x + \theta)]f(x) dx \\ &= \exp(-\theta s) \int_{x=0}^{x=\infty} \exp(-sx)f(x) dx = \exp(-\theta s)L\{f(t)\} = \exp(-\theta s)f(s) \end{aligned} \quad (1.90)$$

The unit impulse function (Figure 1.8) is defined as

$$\delta(t - \theta) \equiv \lim_{m \rightarrow 0} f_m(t - \theta) \quad (1.91)$$

$$f_m(t - \theta) \equiv \begin{cases} 1/m & \theta < t \leq \theta + m \\ 0 & \text{otherwise} \end{cases} \quad (1.92)$$

The impulse function has the following Laplace transform:

$$\begin{aligned} L\{\delta(t - \theta)\} &= \int_0^{\infty} \exp(-st)\delta(t - \theta) dt = \lim_{m \rightarrow 0} \int_{\theta}^{\theta + m} \frac{\exp(-st)}{m} dt \\ &= \lim_{m \rightarrow 0} \left[-\frac{\exp(-st)}{sm} \Big|_{\theta}^{\theta + m} \right] = \lim_{m \rightarrow 0} \left[-\frac{\exp(-\theta s)\exp(-sm)}{sm} + \frac{\exp(-\theta s)}{sm} \right] \end{aligned} \quad (1.93)$$

$$L\{\delta(t - \theta)\} = \exp(-\theta s) \lim_{m \rightarrow 0} \left(\frac{1 - \exp(-sm)}{sm} \right) = \exp(-\theta s) \quad (1.94)$$

$$L\{\delta(t)\} = 1 \quad (1.95)$$

Example 1.38

The algebraic equation (1.96) in the Laplace domain is equivalent to the differential equation (1.97) in the time domain:

$$s^2 y(s) + 3s y(s) + y(s) = 2s u(s) \exp(-2s) + u(s) \exp(-2s) \quad (1.96)$$

$$\begin{aligned} \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) &= 2 \frac{du(t-2)}{dt} S(t-2) + u(t-2)S(t-2) \\ \left. \frac{dy(t)}{dt} \right|_{t=0} &= y(0) = 0 \end{aligned} \quad (1.97)$$

Also, note that $2[du(t-2)/dt]S(t-2) + u(t-2)S(t-2)$ is equivalent to $2[du(t-2)/dt] + u(t-2)$ with $u(t)=0$ for $t \leq 0$. So, (1.97) can be rewritten:

$$\begin{aligned} \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) &= 2 \frac{du(t-2)}{dt} + u(t-2) \\ \left. \frac{dy(t)}{dt} \right|_{t=0} &= y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0 \end{aligned} \quad (1.98)$$

Example 1.39

The algebraic equation (1.99) in the Laplace domain is equivalent to the differential equation (1.100) in the time domain:

$$s^2 y(s) + 3s y(s) + y(s) = 2 \exp(-3s) \quad (1.99)$$

$$\frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = 2\delta(t-3) \quad (1.100)$$

1.3.4 Differentiation and Integration of Laplace Transforms, Convolution

From $L\{f(t)\} = f(s) = \int_0^\infty \exp(-st)f(t)dt$, we obtain

$$\frac{df(s)}{ds} = \int_0^\infty -t \exp(-st)f(t) dt = - \int_0^\infty \exp(-st)tf(t) dt = -L\{tf(t)\}$$

So, the Laplace transform of $tf(t)$ is obtained by differentiating the Laplace transform of $f(t)$ as follows:

$$L\{tf(t)\} = - \frac{df(s)}{ds} \quad (1.101)$$

Meanwhile, the Laplace transform of $f(t)/t$ can be obtained by integrating the Laplace transform of $f(t)$:

$$\begin{aligned} \int_s^\infty f(\hat{s}) d\hat{s} &= \int_s^\infty \left\{ \int_0^\infty \exp(-\hat{s}t)f(t) dt \right\} d\hat{s} = \int_0^\infty \left\{ \int_s^\infty \exp(-\hat{s}t)f(t) d\hat{s} \right\} dt \\ &= \int_0^\infty f(t) \left\{ \int_s^\infty \exp(-\hat{s}t) d\hat{s} \right\} dt = \int_0^\infty f(t) \left\{ - \frac{\exp(-\hat{s}t)}{t} \Big|_s^\infty \right\} dt \\ &= \int_0^\infty \exp(-st) \frac{f(t)}{t} dt = L\left\{ \frac{f(t)}{t} \right\} \end{aligned}$$

Then:

$$L\left\{\frac{f(t)}{t}\right\} = \int_s^\infty f(\hat{s}) \, d\hat{s} \quad (1.102)$$

Consider the following equation to derive the convolution theorem:

$$\begin{aligned} f(s)g(s) &= \int_0^\infty \exp(-s\tau)f(\tau)g(s) \, d\tau = \int_0^\infty f(\tau)\exp(-s\tau)g(s) \, d\tau \\ &= \int_0^\infty f(\tau)\exp(-s\tau) \int_0^\infty \exp(-sx)g(x) \, dx \, d\tau \end{aligned} \quad (1.103)$$

Equation (1.103) can be rewritten by using $t = x + \tau$ and changing the order of the integration as follows:

$$\begin{aligned} f(s)g(s) &= \int_0^\infty f(\tau) \int_\tau^\infty \exp(-st)g(t-\tau) \, dt \, d\tau = \int_0^\infty \int_\tau^\infty \exp(-st)f(\tau)g(t-\tau) \, dt \, d\tau \\ &= \int_0^\infty \exp(-st) \left\{ \int_0^t f(\tau)g(t-\tau) \, d\tau \right\} \, dt \end{aligned} \quad (1.104)$$

So, the following convolution theorem is derived:

$$L\{f(t)\}L\{g(t)\} = f(s)g(s) = \int_0^\infty e^{-st} \left\{ \int_0^t f(\tau)g(t-\tau) \, d\tau \right\} \, dt = L\left\{ \int_0^t f(\tau)g(t-\tau) \, d\tau \right\} \quad (1.105)$$

Example 1.40

The Laplace transform of $t\exp(-3t)$ is

$$-\frac{d}{ds} \left(\frac{1}{s+3} \right) = \frac{1}{(s+3)^2}$$

by (1.101).

Example 1.41

The Laplace transform of $\exp(-3t)$ is

$$\int_s^\infty \frac{1}{(s+3)^2} \, ds = \frac{1}{s+3}$$

by (1.102).

Example 1.42

The convolution theorem means that $L\left\{ \int_0^t f(\tau)g(t-\tau) \, d\tau \right\} = f(s)g(s)$. For example, the inverse Laplace transform of $1/[(s+3)(s+1)]$ is

$$\int_0^t \exp(-3\tau)\exp(-t+\tau) \, d\tau = \exp(-t) \int_0^t \exp(-2\tau) \, d\tau = -\frac{\exp(-3t)}{2} + \frac{\exp(-t)}{2}$$

1.3.5 Laplace Transform of Periodic Functions

Assume that $f(t)$ is a periodic function of which the period is p . Then, the followings are valid:

$$\begin{aligned}
 f(s) &= \int_0^{\infty} \exp(-st)f(t) dt = \int_0^p \exp(-st)f(t) dt + \int_p^{2p} \exp(-st)f(t) dt \\
 &+ \int_{2p}^{3p} \exp(-st)f(t) dt + \int_{3p}^{4p} \exp(-st)f(t) dt + \cdots = \int_0^p \exp(-st)f(t) dt \\
 &+ \int_p^{2p} \exp(-st)f(t-p) dt + \int_{2p}^{3p} \exp(-st)f(t-2p) dt + \int_{3p}^{4p} \exp(-st)f(t-3p) dt + \cdots
 \end{aligned} \tag{1.106}$$

Substituting $\tau = t - p$, we find

$$\int_p^{2p} \exp(-st)f(t-p) dt = \exp(-sp) \int_0^p \exp(-s\tau)f(\tau) d\tau$$

Similarly, we obtain

$$\int_{2p}^{3p} \exp(-st)f(t-2p) dt = \exp(-2sp) \int_0^p \exp(-s\tau)f(\tau) d\tau$$

Then:

$$\begin{aligned}
 f(s) &= \int_0^p \exp(-st)f(t) dt + \exp(-sp) \int_0^p \exp(-s\tau)f(\tau) d\tau + \exp(-2sp) \int_0^p \exp(-s\tau)f(\tau) d\tau \\
 &+ \exp(-3sp) \int_0^p \exp(-s\tau)f(\tau) d\tau + \cdots = (1 + \exp(-sp) + \exp(-2sp) + \exp(-3sp) + \cdots) \int_0^p \exp(-st)f(t) dt \\
 &= \frac{1}{1 - \exp(-sp)} \int_0^p \exp(-st)f(t) dt \quad \text{for } s > 0
 \end{aligned} \tag{1.107}$$

So, the Laplace transform of a periodic function can be calculated if the function values of only one period are given.

1.3.6 Taylor Series and Padé Approximation of Time Delay

The Laplace transform of the time delay term $S(t - \theta)$ is $\exp(-\theta s)$. The Taylor series is

$$f(s) = \exp(-\theta s) = 1 + \sum_{i=1}^{\infty} \frac{s^i}{i!} \left. \frac{d^i f(s)}{ds^i} \right|_{s=0} = 1 + \sum_{i=1}^{\infty} \frac{(-\theta)^i s^i}{i!}$$

Then:

$$u(s)\exp(-\theta s) = u(s) + \sum_{i=1}^{\infty} \frac{(-\theta)^i s^i}{i!} u(s)$$

is obtained and

$$u(t - \theta)S(t - \theta) = u(t) + \sum_{i=1}^{\infty} \frac{(-\theta)^i}{i!} \frac{d^i u(t)}{dt^i}$$

equivalently. Now, we realize that the delayed signal can be represented by a linear combination of the derivatives of the original signal.

The approximation of

$$f(s) = \exp(-\theta s) = 1 + \sum_{i=1}^{\infty} \frac{(-\theta)^i s^i}{i!} \approx 1 - \theta s$$

is called the Taylor series approximation and

$$f(s) = \exp(-\theta s) = \frac{\exp(-0.5\theta s)}{\exp(0.5\theta s)} \approx \frac{1 - \theta s/2}{1 + \theta s/2}$$

is called the Padé approximation. This means that the delayed signal of $g(t) = u(t - \theta)S(t - \theta)$ can be approximated like

$$g(t) = u(t - \theta)S(t - \theta) \approx u(t) - \theta \frac{du(t)}{dt}$$

or

$$g(t) + \frac{\theta}{2} \frac{dg(t)}{dt} \approx u(t) - \frac{\theta}{2} \frac{du(t)}{dt}$$

1.3.7 Partial Fractions

Partial fractions are very useful to restore the original function in the time domain from the Laplace transform in the Laplace domain. Consider the following examples.

Example 1.43

Obtain the inverse Laplace transform for the following Laplace transform:

$$y(s) = \frac{s+2}{s(s+1)(s-2)} \quad (1.108)$$

Solution $y(s)$ can be rewritten in the following form if the roots of the denominator are not repeated:

$$y(s) = \frac{s+2}{s(s+1)(s-2)} = \frac{a}{s} + \frac{b}{s+1} + \frac{c}{s-2} \quad (1.109)$$

$a = -1$ is obtained by putting $s = 0$ in (1.110) after multiplying by s on both sides of (1.109):

$$\frac{s+2}{(s+1)(s-2)} = a + \frac{sb}{s+1} + \frac{sc}{s-2} \quad (1.110)$$

Similarly, $b = 1/3$ by putting $s = -1$ in (1.111) after multiplying by $s + 1$ on both sides of (1.109):

$$\frac{s+2}{s(s-2)} = \frac{(s+1)a}{s} + b + \frac{(s+1)c}{s+3} \quad (1.111)$$

Similarly, $c = 2/3$ by putting $s = 2$ in (1.112) after multiplying by $s - 2$ on both sides of (1.109):

$$\frac{(s+2)}{s(s+1)} = \frac{(s-2)a}{s} + \frac{(s-2)b}{s+1} + c \quad (1.112)$$

Then:

$$y(s) = \frac{s+2}{s(s+1)(s-2)} = \frac{-1}{s} + \frac{1/3}{s+1} + \frac{2/3}{s-2}, \quad y(t) = -1 + \frac{1}{3}\exp(-t) + \frac{2}{3}\exp(2t) \quad (1.113)$$

Example 1.44

Obtain the inverse Laplace transform for the following Laplace transform:

$$y(s) = \frac{s^3 - 4s^2 + 4}{s^2(s-2)(s-1)} \quad (1.114)$$

Solution $y(s)$ can be rewritten in the following form if some of the roots of the denominator are repeated:

$$y(s) = \frac{s^3 - 4s^2 + 4}{s^2(s-2)(s-1)} = \frac{a_2}{s^2} + \frac{a_1}{s} + \frac{b}{s-2} + \frac{c}{s-1} \quad (1.115)$$

$b = -1$ is obtained by putting $s = 2$ after multiplying by $s - 2$ on both sides of (1.115). In a similar way, $c = -1$ and $a_2 = 2$ can also be obtained.

To obtain a_1 , multiply by s^2 on both sides and differentiate both sides with respect to s . Finally, put $s = 0$.

$$\frac{s^3 - 4s^2 + 4}{(s-2)(s-1)} = a_2 + sa_1 + \frac{s^2b}{s-2} + \frac{s^2c}{s-1} \quad (1.116)$$

$$\frac{d}{ds} \left[\frac{s^3 - 4s^2 + 4}{(s-2)(s-1)} \right]_{s=0} = a_1 + \frac{d}{ds} \left(\frac{s^2b}{s-2} \right)_{s=0} + \frac{d}{ds} \left(\frac{s^2c}{s-1} \right)_{s=0} \quad (1.117)$$

where it is noted that

$$\frac{d}{ds} \left(\frac{s^2b}{s-2} \right)_{s=0} = \frac{d}{ds} \left(\frac{s^2c}{s-1} \right)_{s=0} = 0$$

because they include s^2 . So, the following simple equation is obtained:

$$\frac{d}{ds} \left[\frac{s^3 - 4s^2 + 4}{(s-2)(s-1)} \right]_{s=0} = 3 = a_1 \quad (1.118)$$

Finally, the following inverse Laplace transform is obtained:

$$y(s) = \frac{s^3 - 4s^2 + 4}{s^2(s-2)(s-1)} = \frac{2}{s^2} + \frac{3}{s} - \frac{1}{s-2} - \frac{1}{s-1}, \quad y(t) = 2t + 3 - \exp(2t) - \exp(t) \quad (1.119)$$

Example 1.45

Obtain the inverse Laplace transform for the following Laplace transform:

$$y(s) = \frac{2}{(s^2 + 1)(s + 1)^3} \quad (1.120)$$

Solution $y(s)$ can be rewritten in the following form if the complex roots of the denominator are not repeated:

$$y(s) = \frac{2}{(s^2 + 1)(s + 1)^3} = \frac{a_1 s + a_2}{s^2 + 1} + \frac{b_3}{(s + 1)^3} + \frac{b_2}{(s + 1)^2} + \frac{b_1}{(s + 1)} \quad (1.121)$$

As discussed in the Example 1.43, multiply $(s + 1)^3$ on both sides as shown in (1.122) and put $s = -1$ to obtain $b_3 = 1$. As discussed in Example 1.44, multiply by $(s + 1)^3$ on both sides and differentiate both sides with respect to s , as shown in (1.123), and put $s = -1$ to obtain $b_2 = 1$.

$$\frac{2}{s^2 + 1} = \frac{(s + 1)^3(a_1 s + a_2)}{s^2 + 1} + b_3 + b_2(s + 1) + b_1(s + 1)^2 \quad (1.122)$$

$$\begin{aligned} \frac{d}{ds} \left(\frac{2}{s^2 + 1} \right) \Big|_{s=-1} &= \frac{d}{ds} \left[\frac{(s + 1)^3(a_1 s + a_2)}{s^2 + 1} \right] \Big|_{s=-1} + b_2 \frac{d}{ds} (s + 1) \Big|_{s=-1} \\ &+ b_1 \frac{d}{ds} (s + 1)^2 \Big|_{s=-1} = b_2 \end{aligned} \quad (1.123)$$

To obtain b_1 , multiply $(s + 1)^3$ on both sides and differentiate both sides twice with respect to s , as shown in (1.124). Finally, put $s = -1$ and obtain $b_1 = 1/2$.

$$\frac{d^2}{ds^2} \left(\frac{2}{s^2 + 1} \right) \Big|_{s=-1} = \frac{d^2}{ds^2} \left[\frac{(s + 1)^3(a_1 s + a_2)}{s^2 + 1} \right] \Big|_{s=-1} + b_1 \frac{d^2}{ds^2} (s + 1)^2 \Big|_{s=-1} = 2b_1 \quad (1.124)$$

Then, we obtain

$$y(s) = \frac{2}{(s^2 + 1)(s + 1)^3} = \frac{a_1 s + a_2}{s^2 + 1} + \frac{1}{(s + 1)^3} + \frac{1}{(s + 1)^2} + \frac{1/2}{(s + 1)}$$

Equivalently:

$$2 = (a_1 s + a_2)(s + 1)^3 + (s^2 + 1) + (s^2 + 1)(s + 1) + \frac{(s^2 + 1)(s + 1)^2}{2} \quad (1.125)$$

The following equations are obtained by comparing the two terms corresponding to the fourth and third order.

$$s^4 : \quad 0 = 1/2 + a_1 \quad (1.126)$$

$$s^3 : \quad 0 = 1 + 1 + 3a_1 + a_2 \quad (1.127)$$

Therefore:

$$y(s) = \frac{2}{(s^2 + 1)(s + 1)^3} = \frac{-1/2s - 1/2}{s^2 + 1} + \frac{1}{(s + 1)^3} + \frac{1}{(s + 1)^2} + \frac{1/2}{s + 1} \quad (1.128)$$

$$y(t) = -\frac{\cos(t)}{2} - \frac{\sin(t)}{2} + t^2 \frac{\exp(-t)}{2} + t \exp(-t) + \frac{\exp(-t)}{2} \quad (1.129)$$

1.3.8 Solving Differential Equations

The following examples demonstrate how the Laplace transform can be applied to solve differential equations. Note that the Laplace transform is used to convert differential equations in the time domain to algebraic equations in the Laplace domain. It is relatively easy to obtain solutions in the form of the Laplace transform by solving the algebraic equations. Finally, the solution in the time domain can be obtained with the inverse Laplace transform of the solution obtained in the form of the Laplace transform.

Example 1.46

Let us solve the following differential equations using the Laplace transform:

$$\frac{dy(t)}{dt} = -y(t) + 1, \quad y(0) = 0.5 \quad (1.130)$$

Equation (1.131) is obtained by applying the Laplace transform to (1.130):

$$sy(s) - 0.5 = -y(s) + \frac{1}{s} \quad (1.131)$$

Equation (1.131) is rewritten as

$$y(s) = \frac{0.5}{s+1} + \frac{1}{s(s+1)} = -\frac{0.5}{s+1} + \frac{1}{s} \quad (1.132)$$

Then, the ultimate solution in the time domain is obtained by applying the inverse Laplace transform to (1.132):

$$y(t) = 1 - 0.5\exp(-t) \quad (1.133)$$

Example 1.47

Let us solve the following differential equations using the Laplace transform:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = S(t-0.5), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0 \quad (1.134)$$

The Laplace transform of (1.134) is

$$s^2y(s) + 2sy(s) + y(s) = \frac{\exp(-0.5s)}{s} \quad (1.135)$$

Equation (1.135) is rewritten as

$$y(s) = \frac{\exp(-0.5s)}{s(s+1)^2} = \left[\frac{1}{s} - \frac{1}{(s+1)^2} - \frac{1}{s+1} \right] \exp(-0.5s) \quad (1.136)$$

Then, the inverse Laplace transform of (1.136) results in the following solution:

$$y(t) = \{1 - \exp[-(t-0.5)] - (t-0.5)\exp[-(t-0.5)]\}S(t-0.5) \quad (1.137)$$

Equation (1.137) is equivalent to the following representation:

$$y(t) = \begin{cases} 1 - \exp[-(t-0.5)] - (t-0.5)\exp[-(t-0.5)] & t > 0.5 \\ 0 & t \leq 0.5 \end{cases} \quad (1.138)$$

Example 1.48

Let us solve the following differential equations using the Laplace transform:

$$\frac{dy_1(t)}{dt} = -y_1(t) + 2y_2(t) + 1 \quad (1.139)$$

$$\frac{dy_2(t)}{dt} = 2y_1(t) + 2y_2(t) \quad (1.140)$$

$$y_1(0) = 0, \quad y_2(0) = 0.5 \quad (1.141)$$

The following algebraic equations (1.142)–(1.145) in the Laplace domain are obtained by applying the Laplace transform to the differential equations. Equations (1.146) and (1.147) are the solutions obtained by solving the algebraic equations (1.144) and (1.145). Equations (1.148) and (1.149) are the ultimate solutions in the time domain, found by the inverse Laplace transform of the algebraic equations (1.146) and (1.147).

$$sy_1(s) - y_1(0) = -y_1(s) + 2y_2(s) + \frac{1}{s} \quad (1.142)$$

$$sy_2(s) - y_2(0) = 2y_1(s) + 2y_2(s) \quad (1.143)$$

$$(s+1)y_1(s) - 2y_2(s) = \frac{1}{s} \quad (1.144)$$

$$-2y_1(s) + (s-2)y_2(s) = 0.5 \quad (1.145)$$

$$y_1(s) = \frac{s-2}{s(s-3)(s+2)} + \frac{1}{(s-3)(s+2)} = \frac{1/3}{s} + \frac{4/15}{s-3} - \frac{3/5}{s+2} \quad (1.146)$$

$$y_2(s) = \frac{2}{s(s-3)(s+2)} + \frac{1/2(s+1)}{(s-3)(s+2)} = \frac{-1/3}{s} + \frac{8/15}{s-3} + \frac{3/10}{s+2} \quad (1.147)$$

$$y_1(t) = \frac{1}{3} + \frac{4\exp(3t)}{15} - \frac{3\exp(-2t)}{5} \quad (1.148)$$

$$y_2(t) = -\frac{1}{3} + \frac{8\exp(3t)}{15} + \frac{3\exp(-2t)}{10} \quad (1.149)$$

1.3.9 Relationship between Laplace Domain and Time Domain

A lot of the literature on process dynamics and control uses the Laplace transform to explain the core algorithms and signal flows. On the other hand, there is a need to understand the algorithms and signal flows in the form of differential equations to simulate or implement them. Thus, it is important to understand exactly the relationship between the Laplace domain and the time domain. If the formula derived in this chapter is used, then the algebraic equations in the Laplace domain can be obtained from the differential equations in the time domain, and vice versa. Refer to the following examples.

Example 1.49

The equation

$$u(s) = 1.5e(s) \quad (1.150)$$

is equivalent to

$$u(t) = 1.5e(t) \quad (1.151)$$

Example 1.50

The equation

$$y(s) = \frac{(5s+4)\exp(-0.1s)}{3s^2+2s+1}u(s) \quad (1.152)$$

is equivalent to the following differential equations:

$$3 \frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = 5 \frac{du(t-0.1)}{dt} + 4u(t-0.1) \quad (1.153)$$

$$\left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0 \quad (1.154)$$

Example 1.51

The equation

$$u(s) = 1.2 \left(1 + \frac{1}{10.0s} + 2.0s \right) e(s) \quad (1.155)$$

is equivalent to the following differential equation:

$$u(t) = 1.2e(t) + \frac{1.2}{10.0} \int_0^t e(\tau) d\tau + 1.2 \times 2.0 \frac{de(t)}{dt}, \quad e(0) = 0 \quad (1.156)$$

Example 1.52

The multivariable equation

$$u(s) = 1.2 \left(1 + \frac{1}{10.0s} \right) e_1(s) + 0.2 \left(1 + \frac{1}{30.0s} \right) e_2(s) \quad (1.157)$$

is equivalent to the following differential equation:

$$u(t) = 1.2e_1(t) + \frac{1.2}{10.0} \int_0^t e_1(\tau) d\tau + 0.2e_2(t) + \frac{0.2}{30.0} \int_0^t e_2(\tau) d\tau \quad (1.158)$$

1.4 Transfer Function and State-Space Systems

The transfer function is a simple and useful tool to represent the relationship between the process input and the process output. The state-space representation is required to simulate the dynamics of the given transfer function.

1.4.1 Transfer Function

The transfer function from $u(t)$ to $y(t)$ is defined as the Laplace transform of $y(t)$ over the Laplace transform of $u(t)$ assuming that the initial values are zero and steady state (denoted by the term initial zero-steady-state).

$$G(s) = \frac{y(s)}{u(s)} \quad (1.159)$$

If $u(t)$ and the other variables simultaneously affect $y(t)$, then the other variables should be assumed zeroes to obtain the transfer function from $u(t)$ to $y(t)$. Refer to the following examples.

Example 1.53

Obtain the transfer function from $u(t)$ to $y(t)$ for the process

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 3\frac{du(t)}{dt} + u(t) \quad (1.160)$$

Solution Assuming the initial zero-steady-state (that is, $dy(t)/dt|_{t=0} = y(0) = 0$, $u(0) = 0$), the Laplace transform (1.161) and the transfer function (1.162) are obtained:

$$s^2y(s) + 2sy(s) + y(s) = 3su(s) + u(s) \quad (1.161)$$

$$\frac{y(s)}{u(s)} = \frac{3s + 1}{s^2 + 2s + 1} \quad (1.162)$$

Example 1.54

Obtain the transfer function from $u(t)$ to $y(t)$ for the process

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 3\frac{du(t-2)}{dt} + u(t-2) \quad (1.163)$$

Solution Assuming the initial zero-steady-state (that is, $dy(t)/dt|_{t=0} = y(0) = 0$, $u(t) = 0$ for $t < 0$), the following Laplace transform (1.164) and transfer function (1.165) are obtained:

$$s^2y(s) + 2sy(s) + y(s) = 3su(s)\exp(-2s) + u(s)\exp(-2s) \quad (1.164)$$

$$\frac{y(s)}{u(s)} = \frac{(3s + 1)\exp(-2s)}{s^2 + 2s + 1} \quad (1.165)$$

Example 1.55

Obtain the transfer function from $u(t)$ to $y(t)$ and the transfer function from $v(t)$ to $y(t)$ for the following process, noting that $u(t)$ and $v(t)$ simultaneously affect $y(t)$:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 3\frac{du(t-2)}{dt} + u(t-2) + 2v(t-1) \quad (1.166)$$

Solution To obtain the transfer function from $u(t)$ to $y(t)$, assume the initial zero-steady-state (that is, $dy(t)/dt|_{t=0} = y(0) = 0$, $u(t) = 0$ for $t \leq 0$) and the variable $v(t) = 0$. Then, the Laplace transform (1.167) and the transfer function from $u(t)$ to $y(t)$ (1.168) are obtained:

$$s^2y(s) + 2sy(s) + y(s) = 3su(s)\exp(-2s) + u(s)\exp(-2s) \quad (1.167)$$

$$\frac{y(s)}{u(s)} = \frac{(3s+1)\exp(-2s)}{s^2 + 2s + 1} \quad (1.168)$$

To obtain the transfer function from $v(t)$ to $y(t)$, assume the initial zero-steady-state (that is, $dy(t)/dt|_{t=0} = y(0) = 0$, $v(t) = 0$ for $t \leq 0$) and the variable $u(t) = 0$. Then, the Laplace transform (1.169) and the transfer function from $v(t)$ to $y(t)$ (1.170) are obtained:

$$s^2y(s) + 2sy(s) + y(s) = 2v(s)\exp(-s) \quad (1.169)$$

$$\frac{y(s)}{v(s)} = \frac{2\exp(-s)}{s^2 + 2s + 1} \quad (1.170)$$

Remark Equation (1.166) can be rewritten as follows using the superposition rule:

$$\frac{d^2y_u(t)}{dt^2} + 2\frac{dy_u(t)}{dt} + y_u(t) = 3\frac{du(t-2)}{dt} + u(t-2) \quad (1.171)$$

$$\frac{d^2y_v(t)}{dt^2} + 2\frac{dy_v(t)}{dt} + y_v(t) = 2v(t-1) \quad (1.172)$$

$$y(t) = y_u(t) + y_v(t) \quad (1.173)$$

The transfer function from $u(t)$ to $y(t)$ corresponds to the transfer function from $u(t)$ to $y_u(t)$ and the transfer function from $v(t)$ to $y(t)$ corresponds to the transfer function from $v(t)$ to $y_v(t)$. Now, it is clear from (1.173) that the relationship between the process output $y(s)$ and the process inputs $u(s)$ and $v(s)$ can be obtained by adding the two transfer functions as follows:

$$y(s) = \frac{(3s+1)\exp(-2s)}{s^2 + 2s + 1}u(s) + \frac{2\exp(-s)}{s^2 + 2s + 1}v(s) \quad (1.174)$$

1.4.2 State-Space Systems

The transfer function

$$G(s) = \frac{y(s)}{u(s)} = \frac{b_1s^{n-1} + b_2s^{n-2} + \cdots + b_{n-1}s + b_n}{s^n + a_1s^{n-1} + \cdots + a_{n-1}s + a_n}\exp(-\theta s)$$

is known to be equivalent to the following differential equations with the following initial conditions:

$$\begin{aligned} \frac{d^n y(t)}{dt^n} + a_1 \frac{d^{n-1} y(t)}{dt^{n-1}} + \cdots + a_{n-1} \frac{dy(t)}{dt} + a_n y(t) = b_1 \frac{d^{n-1} u(t-\theta)}{dt^{n-1}} + \cdots \\ + b_{n-1} \frac{du(t-\theta)}{dt} + b_n u(t-\theta) \end{aligned} \quad (1.175)$$

$$u(t) = 0 \quad \text{and} \quad \frac{d^i u(t)}{dt^i} = 0, \quad i = 1, 2, \dots, n-2 \quad \text{for } t \leq 0 \quad (1.176)$$

$$y(t) = 0 \quad \text{and} \quad \frac{d^i y(t)}{dt^i} = 0, \quad i = 1, 2, \dots, n-1 \quad \text{for } t \leq 0 \quad (1.177)$$

It is proven that the differential equation system (1.175)–(1.177) is equivalent to the following state-space system, which is called the state-space representation (realization):

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t-\theta) \quad (1.178)$$

$$y(t) = Cx(t) \quad (1.179)$$

$$x(0) = 0 \quad (1.180)$$

where $u(t-\theta)$ and $y(t)$ denote the delayed process input and the scalar process output respectively. $x(t)$ is the n -dimensional state. Here, the system matrices are as follows:

$$A = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_n \\ 1 & 0 & 0 & \cdots & 0 & -a_{n-1} \\ 0 & 1 & 0 & \cdots & 0 & -a_{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -a_2 \\ 0 & 0 & 0 & \cdots & 1 & -a_1 \end{bmatrix} \quad (1.181)$$

$$B = \begin{bmatrix} b_n & b_{n-1} & b_{n-2} & \cdots & b_2 & b_1 \end{bmatrix}^T \quad (1.182)$$

$$C = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \quad (1.183)$$

Note that differential equation (1.175) cannot be solved directly by the usual ordinary differential equation solvers because the high-order differentials are included. This problem can be solved simply by the state-space representation. That is, the solutions $y(t)$ of the differential equations (1.175)–(1.177) are easily obtained by solving the state-space model (1.181)–(1.183) with the usual ordinary differential equation solvers.

Equation (1.180) is valid for the case that all the initial values of the process input and the process output are zeroes, as shown in (1.176) and (1.177). If the initial values of $u(t - \theta)$ and $y(t)$ are not zeroes, then (1.175) is equivalent to (1.178), (1.179) and (1.181)–(1.183) with the initial values $x(0)$ of (1.184).

$$x(0) = \chi^{-1} \psi \quad (1.184)$$

$$\psi = \begin{bmatrix} y(0) \\ \left. \frac{dy}{dt} \right|_{t=0} - CBu(-\theta) \\ \left. \frac{d^2y}{dt^2} \right|_{t=0} - CABu(-\theta) - CB \left. \frac{du}{dt} \right|_{t=-\theta} \\ \vdots \\ \left. \frac{d^{n-1}y}{dt^{n-1}} \right|_{t=0} - CA^{n-2}Bu(-\theta) - CA^{n-3}B \left. \frac{du}{dt} \right|_{t=-\theta} - \cdots - CB \left. \frac{d^{n-2}u}{dt^{n-2}} \right|_{t=-\theta} \end{bmatrix} \quad (1.185)$$

$$\chi = \begin{bmatrix} C \\ CA \\ CA^2 \\ \vdots \\ CA^{n-1} \end{bmatrix} \quad (1.186)$$

Example 1.56

Obtain the state-space process corresponding to the following process:

$$\frac{d^2y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = 3 \frac{du(t-2)}{dt} + u(t-2) \quad (1.187)$$

$$\left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0 \quad (1.188)$$

Solution From (1.178)–(1.183), the state-space process is

$$\frac{dx(t)}{dt} = \begin{bmatrix} 0 & -1 \\ 1 & -2 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ 3 \end{bmatrix} u(t-2) \quad (1.189)$$

$$y(t) = [0 \quad 1] x(t) \quad (1.190)$$

$$x(0) = [0 \quad 0]^T \quad (1.191)$$

Example 1.57

Obtain the state-space process corresponding to the process

$$2\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + 6y(t) = 2u(t-2) \quad (1.192)$$

$$\left.\frac{dy(t)}{dt}\right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0 \quad (1.193)$$

Solution Note that (1.192) should be converted to the standard form (1.175) by dividing (1.192) by 2.0. Then, from (1.178)–(1.183), the state-space process is

$$\frac{dx(t)}{dt} = \begin{bmatrix} 0 & -6/2 \\ 1 & -2/2 \end{bmatrix} x(t) + \begin{bmatrix} 2/2 \\ 0 \end{bmatrix} u(t-2) \quad (1.194)$$

$$y(t) = [0 \quad 1]x(t) \quad (1.195)$$

$$x(0) = [0 \quad 0]^T \quad (1.196)$$

Example 1.58

Obtain the state-space process corresponding to the process

$$\frac{d^3y(t)}{dt^3} + 3.5\frac{d^2y(t)}{dt^2} + 3.2\frac{dy(t)}{dt} + y(t) = -2\frac{du(t-0.5)}{dt} + u(t-0.5) \quad (1.197)$$

$$\left.\frac{d^2y(t)}{dt^2}\right|_{t=0} = \left.\frac{dy(t)}{dt}\right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0 \quad (1.198)$$

Solution From (1.178)–(1.183), the state-space process is

$$\frac{dx(t)}{dt} = \begin{bmatrix} 0 & 0 & -1.0 \\ 1 & 0 & -3.2 \\ 0 & 1 & -3.5 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} u(t-0.5) \quad (1.199)$$

$$y(t) = [0 \quad 0 \quad 1]x(t) \quad (1.200)$$

$$x(0) = [0 \quad 0 \quad 0]^T \quad (1.201)$$

Example 1.59

Obtain the state-space process corresponding to the process

$$\frac{d^3y(t)}{dt^3} + 3.5 \frac{d^2y(t)}{dt^2} + 3.2 \frac{dy(t)}{dt} + y(t) = -2 \frac{du(t-0.5)}{dt} + u(t-0.5) \quad (1.202)$$

$$\left. \frac{d^2y(t)}{dt^2} \right|_{t=0} = 0.1, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = -0.3, \quad y(0) = 0.5, \quad u(-0.5) = 1 \quad (1.203)$$

Solution From (1.178), (1.179), (1.181)–(1.183) and (1.184)–(1.186), the state-space process is

$$\frac{dx(t)}{dt} = \begin{bmatrix} 0 & 0 & -1.0 \\ 1 & 0 & -3.2 \\ 0 & 1 & -3.5 \end{bmatrix} x(t) + \begin{bmatrix} 1 \\ -2 \\ 0 \end{bmatrix} u(t-0.5) \quad (1.204)$$

$$y(t) = [0 \quad 0 \quad 1]x(t) \quad (1.205)$$

$$x(0) = \chi^{-1}\psi \quad (1.206)$$

$$\psi = \begin{bmatrix} 0.5 \\ -0.3 \\ 2.1 \end{bmatrix} \quad (1.207)$$

$$\chi = \begin{bmatrix} 0 & 0 & 1 \\ 0 & 1 & -3.5 \\ 1 & -3.5 & 9.05 \end{bmatrix} \quad (1.208)$$

Problems

- 1.1 Figure P1.1 shows an aeration tank for wastewater treatment. Microorganisms in the tank remove the pollutants in wastewater. The dissolved oxygen (DO) in the wastewater should be controlled to a desired level to maximize the treatment performance of the microorganisms. The blower is to adjust the air flow-rate, which is proportional to the voltage $v(t)$. DO is measured by the DO sensor. Determine the manipulating variable and the controlled variable for the DO control system.
- 1.2 Figure P1.2 shows a schematic diagram for a level control. The valve is to adjust the flow rate and the DP cell measures the liquid level in the tank. Determine the manipulating variable and the controlled variable.
- 1.3 The process output $y(t) = 0.5 - 0.5(1 + t/2) \exp(-t/2)$ is obtained for the process input $u(t) = 1.0$. Estimate the parameters of the following process:

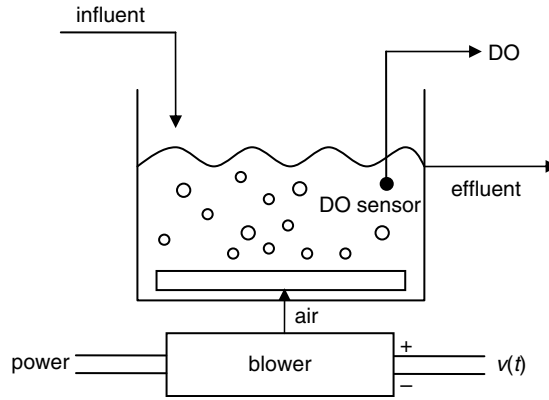


Figure P1.1

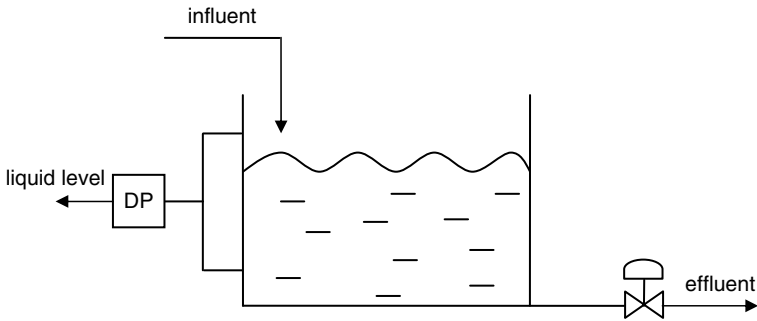


Figure P1.2

$$\tau^2 \frac{d^2 y(t)}{dt^2} + 2\tau \frac{dy(t)}{dt} + y(t) = ku(t)$$

- 1.4 The process output $y(t) = (1/\sqrt{2})\sin(2t - 3\pi/4)$ is obtained for the process input $u(t) = \sin(2t)$. Estimate the parameters of the following process:

$$\tau^3 \frac{d^3 y(t)}{dt^3} + 3\tau^2 \frac{d^2 y(t)}{dt^2} + 3\tau \frac{dy(t)}{dt} + y(t) = ku(t)$$

- 1.5 Obtain $y(t)$ at steady state for the following system:

$$\frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} (2.0 + 0.1u(t)) + y(t) = 0.1 \frac{du(t)}{dt} + \sqrt{u(t)}$$

$$u(t) = 0.5(1 - y(t))$$

1.6 Obtain $y(t)$ at steady state for the process

$$2\frac{dy(t)}{dt} + y(t) = (1 + 0.1y(t))u(t)$$

$$u(t) = 0.5(1 - y(t)) + 0.25\frac{d(1 - y(t))}{dt}$$

1.7 Rewrite the following processes using deviation variables of which the reference value is $y_{\text{ref}} = 0.0$:

- (a) $\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) + 3 = 0.5\frac{du(t)}{dt} - u(t)$
- (b) $\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) - 0.1y(t - 0.1) = 0.5\frac{du(t - 0.2)}{dt} - u(t - 0.2) + 5$

1.8 Choose linear processes and determine if they are time invariant or time variant.

- (a) $\frac{d^2y(t)}{dt^2} + \frac{dy(t)}{dt} + y(t)(1 + 0.5u(t)) = 0.5\frac{du(t)}{dt} + 2u(t)$
- (b) $(1 + 0.1t)\frac{dy(t)}{dt} + y(t) = (3.0 + t)u(t)$
- (c) $(1 + 0.1t)\frac{dy(t)}{dt} + y(t) = (3.0 + t)u(t - 0.5)$
- (d) $2\frac{dy(t)}{dt} + y(t) = 2u(t), \quad u(t) = 0.5(1 - y(t))$
- (e) $2\frac{dy(t)}{dt} + y(t) = 2u(t - 0.1), \quad u(t) = 0.5(1 - y(t))$
- (f) $2\frac{dy(t)}{dt} + y(t) = 2u(t - 0.1), \quad u(t) = 0.5(1 - y(t)) + 0.2\int_0^t (1 - y(\tau)) d\tau$
- (g) $2\frac{dy(t)}{dt} + y(t) + 3 = 2u(t), \quad u(t) = 0.5(1 - y(t))$
- (h) $\frac{dy(t)}{dt} + y(t) = 1.0$

1.9 Obtain the process outputs $y(0.0), y(0.1), y(0.2), y(0.3), y(0.4)$ of the linear processes for the following respective process inputs. The available information is that the responses of the process to the process input $u_1(t) = 1$ for $t \geq 0$, $u_1(t) = 0$ for $t < 0$ are $y_1(t) = 0.0$ for $t \leq 0.0$, $y_1(0.1) = 0.01$, $y_1(0.2) = 0.02$, $y_1(0.3) = 0.04$, and $y_1(0.4) = 0.07$.

- (a) $u(t) = 3.0$ for $t \geq 0$, $u(t) = 0$ for $t < 0$
- (b) $u(t) = -2.0$ for $t \geq 0$, $u(t) = 0$ for $t < 0$
- (c) $u(t) = 1.0$ for $t \geq 0.2$, $u(t) = 3.0$ for $0 \leq t < 0.2$, $u(t) = 0$ for $t < 0$
- (d) $u(t) = -1.0$ for $t \geq 0.2$, $u(t) = 0$ for $t < 0.2$
- (e) $u(t) = -1.0$ for $t \geq 0.2$, $u(t) = 0$ for $0.1 \leq t < 0.2$, $u(t) = 2.0$ for $0 \leq t < 0.1$, $u(t) = 0$ for $t < 0$

1.10 Obtain the process outputs $y(t)$ of the linear process for each of the following process inputs. The available information is that the responses of the process to the process input $u_1(t) = \sin(t) + \sin(2t) + 1$ are $y_1(t) = 0.7 \sin(t - 0.1) + 0.5 \sin(2t - 0.2) + 1.2$

- (a) $u(t) = 2 \sin(t)$
- (b) $u(t) = 2 \sin(2t)$
- (c) $u(t) = -2.0$
- (d) $u(t) = 3 \sin(t) + 2 \sin(2t)$
- (e) $u(t) = 2 \sin(t) + 2.0$
- (f) $u(t) = 2 \sin(t - 0.2)$
- (g) $u(t) = 4.0 \sin(t - 0.2) + 3.0 \sin(2t - 0.1) + 2.0$

1.11 Obtain the linearized process around $u = 1.0$ for the following nonlinear processes and express it with the deviation variables:

- (a) $\frac{dy(t)}{dt} + y(t)(1 + 0.1u^2(t)) = \sqrt{u(t)}$
- (b) $\frac{d^2y(t)}{dt^2} + y^3(t) + 0.05 \frac{dy(t)}{dt} u(t) = u(t)$
- (c) $\frac{dy(t)}{dt} + y^3(t) = u(t)$
- (d) $\frac{d^2y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + 0.05 \left(\frac{dy(t)}{dt} \right)^2 y^2(t) u(t) + y(t) = u^3(t)$
- (e) $\frac{dy(t)}{dt} + y^3(t) + 5 = u(t) + 2$

1.12 Find $y(t)$ for the following Laplace transforms:

- (a) $y(s) = \frac{2}{(2s+1)s}$
- (b) $y(s) = \frac{3}{s(s-2)(s+2)}$
- (c) $y(s) = \frac{1}{(s+1)^2 s}$
- (d) $y(s) = \frac{1}{(s+1)^3 s}$
- (e) $y(s) = \frac{\exp(-0.5s)}{(s+1)^3 s}$
- (f) $y(s) = \frac{1}{(s+1)^2 s} - \frac{\exp(-0.5s)}{(s+1)^3 s}$
- (g) $y(s) = \frac{1}{s^3(s-2)(s+2)}$

1.13 Find $y(t)$ for the following differential equations using the Laplace transform:

$$(a) \quad \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + 2y(t) = 0, \quad y(0) = 1, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = 0$$

$$(b) \quad \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + 2y(t) = 1, \quad y(0) = \left. \frac{dy(t)}{dt} \right|_{t=0} = 0$$

$$(c) \quad \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + 2y(t) = u(t), \quad u(t) = 1.0 \quad \text{for} \\ t \geq 0.5, \quad u(t) = 0.0 \quad \text{for } t < 0.5 \text{ and } y(0) = \left. \frac{dy(t)}{dt} \right|_{t=0} = 0$$

$$(d) \quad \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + 2y(t) = u(t), \quad u(t) = 1.0 \quad \text{for } 0.5 \leq t < 1.0, \quad u(t) = 0.0 \quad \text{for} \\ t < 0.5 \text{ or } t \geq 1.0, \text{ and } y(0) = \left. \frac{dy(t)}{dt} \right|_{t=0} = 0$$

1.14 Find $y_1(t)$, $y_2(t)$ and $y_3(t)$ for the following differential equations using the Laplace transform:

$$\frac{dy_1(t)}{dt} = -y_3(t) + 1.0$$

$$\frac{dy_2(t)}{dt} = y_1(t) - 3.0y_3(t) - 0.5$$

$$\frac{dy_3(t)}{dt} = y_2(t) - 3.0y_3(t)$$

$$y_1(0) = y_2(0) = y_3(0) = 0$$

1.15 Rewrite the following differential equations into the algebraic equations in the Laplace domain:

$$(a) \quad 2 \frac{dy(t)}{dt} + y(t) = 3.0(1 - y(t)), \quad y(0) = 0$$

$$(b) \quad 2 \frac{dy(t)}{dt} + y(t) = 3.0(1 - y(t)) + 1.0 \int_0^t (1 - y(\tau)) d\tau, \quad y(0) = 0$$

$$(c) \quad 2 \frac{dy(t)}{dt} + y(t) = 3.0(1 - y(t)) + 1.0 \int_0^t (1 - y(\tau)) d\tau + 0.5 \frac{d(1 - y(t))}{dt}, \quad y(0) = 0$$

$$(d) \quad \frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = 3.0(1 - y(t)), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0$$

$$(e) \quad 2 \frac{dy(t)}{dt} + y(t) = 3.0(S(t - 0.5) - y(t - 0.5)S(t - 0.5)), \quad y(t) = 0 \quad \text{for } t \leq 0$$

$$(f) \quad 2 \frac{dy(t)}{dt} + y(t) = 1.0 \int_0^t [S(\tau - 0.5) - y(\tau - 0.5)] d\tau + 0.5 \frac{d[S(t - 0.5) - y(t - 0.5)]}{dt},$$

$$y(t) = 0 \text{ for } t \leq 0$$

$$(g) \quad 2 \frac{dy(t)}{dt} + y(t) = u(t - 0.5)S(t - 0.5), \quad u(t) = -y(t) + 3.0(1 - y(t)), \quad y(0) = 0$$

1.16 Rewrite the following algebraic equations into the differential equations in the time domain:

$$(a) \quad s^3 y(s) + s^2 y(s) + sy(s) = su(s) + 10u(s)$$

$$(b) \quad s^3 y(s) + s^2 y(s) + sy(s) = (su(s) + 10u(s))\exp(-0.1s)$$

$$(c) \quad u(s) = -\exp(-0.1s)y(s) + \left(\frac{1}{s} - y(s)\right)\exp(-0.1s) + \frac{1}{s}\left(\frac{1}{s} - y(s)\right)\exp(-0.1s) + s\left(\frac{1}{s} - y(s)\right)\exp(-0.1s)$$

$$(d) \quad y(s) = \frac{(-0.1s^2 + 0.5s + 1)\exp(-0.1s)}{s^3 + 3s^2 + s + 1}u(s)$$

$$(e) \quad y(s) = \frac{\exp(-0.1s)}{s^2 + s + 1}u(s) + \frac{\exp(-0.2s)}{2s + 1}v(s)$$

$$(f) \quad y(s) = 5u(s) + 7v(s)$$

1.17 Obtain the transfer function for the following process:

$$(a) \quad 2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} = -0.1 \frac{du(t - 0.5)}{dt} + 3u(t - 0.5), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \text{ for } t \leq 0$$

$$(b) \quad 2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} = 0.5u(t) + 0.1 \int_0^t u(\tau) d\tau + 0.1 \frac{du(t)}{dt}, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \text{ for } t \leq 0$$

$$(c) \quad 2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} = 0.5u(t - 0.5) + 0.1 \int_0^t u(\tau - 0.5) d\tau, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \text{ for } t \leq 0$$

$$(d) \quad 2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} = 0.5u(t - 0.5) + v(t - 0.3), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \text{ for } t \leq 0, \quad v(t) = 0 \text{ for } t \leq 0$$

1.18 Obtain the state-space process corresponding to the following processes:

$$(a) \quad 2 \frac{dy(t)}{dt} + 3y(t) = u(t - 0.5), \quad y(0) = 0, \quad u(t) = 0 \text{ for } t \leq 0$$

$$(b) \quad \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = 0.1 \frac{du(t - 0.5)}{dt} + u(t - 0.5), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \text{ for } t \leq 0$$

$$(c) \quad 3 \frac{d^2 y(t)}{dt^2} + 6 \frac{dy(t)}{dt} + 3y(t) = 2u(t - 0.5), \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0,$$

$$u(t) = 0 \quad \text{for } t \leq 0$$

$$(d) \quad 2 \frac{d^4 y(t)}{dt^4} + 8 \frac{d^3 y(t)}{dt^3} + 12 \frac{d^2 y(t)}{dt^2} + 6 \frac{dy(t)}{dt} + 4y(t) = -0.2 \frac{d^2 u(t)}{dt^2} + 1.0 \frac{du(t)}{dt} + 4u(t),$$

$$\left. \frac{d^3 y(t)}{dt^3} \right|_{t=0} = \left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0$$

$$(e) \quad 2 \frac{d^4 y(t)}{dt^4} + 8 \frac{d^3 y(t)}{dt^3} + 12 \frac{d^2 y(t)}{dt^2} + 6 \frac{dy(t)}{dt} + 4y(t) = -0.2 \frac{d^2 u(t - 0.3)}{dt^2} + 1.0 \frac{du(t - 0.3)}{dt} + 4u(t - 0.3),$$

$$\left. \frac{d^3 y(t)}{dt^3} \right|_{t=0} = 0.3, \quad \left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = -0.2, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = 0.5, \quad y(0) = -0.5,$$

$$\left. \frac{du(t)}{dt} \right|_{t=-0.3} = 0.2, \quad u(-0.3) = -0.4$$

Bibliography

Kreyszig, E. (2006) *Advanced Engineering Mathematics*, John Wiley & Sons, Inc.

Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.

Stephanopoulos, G. (1984) *Chemical Process Control – An Introduction to Theory and Practice*, Prentice-Hall.

2

Simulations

2.1 Simulating Processes Composed of Differential Equations

In this section, the numerical derivative will be briefly introduced, followed by a simple method to solve first-order and high-order differential equations.

2.1.1 Numerical Derivative and Solving First-Order Differential Equations

Consider the following derivative definition:

$$\left. \frac{dy(t)}{dt} \right|_{t=t_0} = \lim_{\Delta t \rightarrow 0} \left\{ \frac{y(t_0 + \Delta t) - y(t_0)}{t_0 + \Delta t - t_0} \right\} = \lim_{\Delta t \rightarrow 0} \left\{ \frac{y(t_0 + \Delta t) - y(t_0)}{\Delta t} \right\} \quad (2.1)$$

It should be noted that the derivative can be approximated by choosing a small Δt instead of $\Delta t \rightarrow 0$ as shown below.

$$\left. \frac{dy(t)}{dt} \right|_{t=t_0} \approx \frac{y(t_0 + \Delta t) - y(t_0)}{\Delta t} \quad (2.2)$$

where Δt is a small value. Equation (2.2) is called a numerical derivative.

The MATLAB program in Table 2.1 is used to calculate the numerical derivative of $y(t) = t^2 + 3t + 1$ at $t = 1$ and compares it with the analytical value. As expected, the numerical derivative value (5.000 009 999 989) is very close to the analytical derivative value (5.000 000 000 000).

If the Δt value is decreased further, then the accuracy of the numerical derivative will be improved. But, there is a practical limitation in improving the accuracy because the round-off error increases as Δt decreases.

The same principle can be applied to solve differential equations. Consider the following differential equation:

$$\frac{dy(t)}{dt} = -y(t) + t, \quad y(0) = 1 \quad (2.3)$$

Table 2.1 MATLAB code for the numerical derivative.

numerical_derivative.m	f.m	command window
<pre> t0=1.0; delta_t=0.00001; y0=f(t0); y_delta=f(t0+delta_t); dy_dt=(y_delta-y0)/ delta_t; Numerical_derivative = dy_dt Analytical_solution = (2*t0+3) </pre>	<pre> function [y]=f(t) y=t^2+3*t+1; end </pre>	<pre> >> numerical_derivative Numerical_derivative = 5.00000999998917 Analytical_solution = 5 </pre>

The derivative $dy(t)/dt$ can be approximated as follows:

$$\frac{dy(t)}{dt} \approx \frac{y(t + \Delta t) - y(t)}{\Delta t} \quad (2.4)$$

Then, (2.3) can be rewritten as follows:

$$\frac{y(t + \Delta t) - y(t)}{\Delta t} = -y(t) + t \quad (2.5)$$

$$y(t + \Delta t) = y(t) + (-y(t) + t)\Delta t \quad (2.6)$$

Now, $y(t)$, $t = \Delta t, 2\Delta t, \dots$, can be obtained for the given initial value $y(0)$ by repeating (2.6). For example:

$$t = 0 \quad \text{and} \quad y(0) = 1 \rightarrow y(\Delta t) = y(0) + (-y(0) + 0)\Delta t \quad (2.7)$$

$$t = \Delta t \rightarrow y(2\Delta t) = y(\Delta t) + (-y(\Delta t) + \Delta t)\Delta t \quad (2.8)$$

$$t = 2\Delta t \rightarrow y(3\Delta t) = y(2\Delta t) + (-y(2\Delta t) + 2\Delta t)\Delta t \quad (2.9)$$

$$t = 3\Delta t \rightarrow y(4\Delta t) = y(3\Delta t) + (-y(3\Delta t) + 3\Delta t)\Delta t \quad (2.10)$$

It is straightforward to apply the same method to the more general case of $dy(t)/dt = g(y(t), t)$. For example:

$$y(t + \Delta t) = y(t) + g(y(t), t)\Delta t \quad (2.11)$$

$$t = 0, \quad y(0) = y_0 \rightarrow y(\Delta t) = y(0) + g(y(0), 0)\Delta t \quad (2.12)$$

$$t = \Delta t \rightarrow y(2\Delta t) = y(\Delta t) + g(y(\Delta t), \Delta t)\Delta t \quad (2.13)$$

$$t = 2\Delta t \rightarrow y(3\Delta t) = y(2\Delta t) + g(y(2\Delta t), 2\Delta t)\Delta t$$

(2.14)

$$t = 3\Delta t \rightarrow y(4\Delta t) = y(3\Delta t) + g(y(3\Delta t), 3\Delta t)\Delta t$$

(2.15)

The above-mentioned method to solve differential equations (2.3) is called the Euler method. The MATLAB code in Table 2.2 is used to calculate $y(t)$, $t = 0.01, 0.02, 0.03, \dots$, for the given initial value $y(0) = 1$; the simulation results are given in Figure 2.1.

Table 2.2 MATLAB code to solve the first-order differential equation using the Euler method.

euler1.m	g1.m
<pre>clear; t=0.0; y=1.0; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; dy_dt=g1(y,t); y=y+dy_dt*delta_t; %Eq. (2.11) t=t+delta_t; end figure(1); plot(t_array,y_array);</pre>	<pre>function [dy_dt]=g1(y,t) dy_dt=-y+t; end</pre>
	<div>command window</div> <div>>> euler1</div>

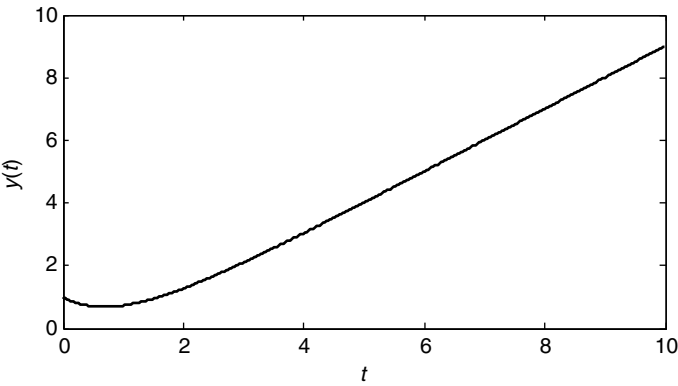


Figure 2.1 Simulation result of Table 2.2.

2.1.2 Solving High-Order Differential Equations

The state-space representation (1.175)–(1.183) should be used to solve a high-order differential equation. For example, consider the following high-order differential equation:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = \frac{d^2 u(t)}{dt^2} - 5 \frac{du(t)}{dt} + u(t) \quad (2.16)$$

$$u(t) = 1 \quad \text{for } t \geq 0 \quad (2.17)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (2.18)$$

To solve the differential equations (2.16)–(2.18), they should be rewritten in the corresponding state-space representation (2.19)–(2.24) according to (1.175)–(1.183):

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t) \quad (2.19)$$

$$y(t) = Cx(t) \quad (2.20)$$

$$x(0) = [0 \quad 0 \quad 0]^T \quad (2.21)$$

$$A = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & -3 \\ 0 & 1 & -3 \end{bmatrix} \quad (2.22)$$

$$B = [1 \quad -5 \quad 1]^T \quad (2.23)$$

$$C = [0 \quad 0 \quad 1] \quad (2.24)$$

Now, it is straightforward to solve the high-order differential equation by repeating the equation

$$x(t + \Delta t) = x(t) + (Ax(t) + Bu(t))\Delta t \quad (2.25)$$

The MATLAB code in Table 2.3 is used to calculate $y(t)$, $t = 0.01, 0.02, 0.03, \dots$, for (2.16)–(2.18) and the simulation results are given in Figure 2.2.

Example 2.1

Obtain the numerical derivative $dy(t)/dt$ at $t = 0.5$ with $\Delta t = 0.01$ for the function

$$y(t) = t \exp(-3t) \quad (2.26)$$

Solution

$$\left. \frac{dy(t)}{dt} \right|_{t=0.5} \approx \frac{y(0.5 + 0.01) - y(0.5)}{0.01} = -0.1132 \quad (2.27)$$

Table 2.3 MATLAB code to solve the high-order differential equation using the Euler method.

euler2.m	g2.m
<pre>clear; t=0.0; x=[0 0 0]'; y=0; C=[0 0 1]; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; u=1.0; dx_dt=g2(y,x,u); x=x+dx_dt*delta_t; %Eq. (2.25) y=C*x; t=t+delta_t; end figure(1); plot(t_array,y_array);</pre>	<pre>function [dx_dt]=g2(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -5 1]'; dx_dt=A*x+B*u; end</pre>
	<pre>command window >> euler2</pre>

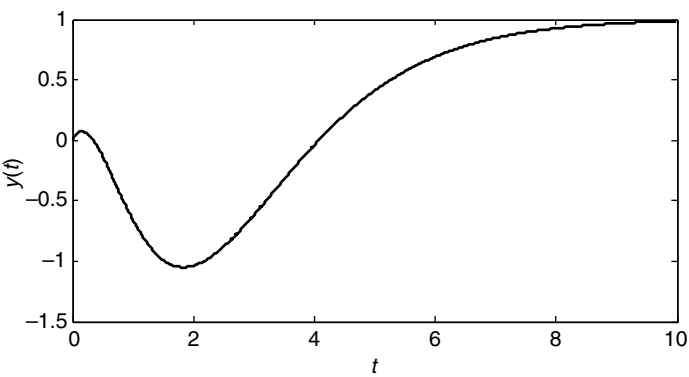


Figure 2.2 Simulation result of Table 2.3.

Example 2.2

Obtain the numerical derivatives $\partial y(x,z)/\partial x$, $\partial^2 y(x,z)/\partial x^2$ and $\partial^2 y(x,z)/\partial x \partial z$ at $x=0.5$ and $z=0.3$ with $\Delta x=0.01$ and $\Delta z=0.005$ for the function

$$y(x,z) = x \exp(-3z) + x^2 + 2z$$

(2.28)

Solution

$$\left. \frac{\partial y(x, z)}{\partial x} \right|_{x=0.5, z=0.3} \approx \frac{y(0.5 + 0.01, 0.3) - y(0.5, 0.3)}{0.01} = 1.4166 \quad (2.29)$$

$$\left. \frac{\partial y(x, z)}{\partial x} \right|_{x=0.5 + 0.01, z=0.3} \approx \frac{y(0.5 + 0.02, 0.3) - y(0.5 + 0.01, 0.3)}{0.01} = 1.4366 \quad (2.30)$$

$$\begin{aligned} \left. \frac{\partial^2 y(x, z)}{\partial x^2} \right|_{x=0.5, z=0.3} &\approx \frac{\left. \frac{\partial y(x, z)}{\partial x} \right|_{x=0.5 + 0.01, z=0.3} - \left. \frac{\partial y(x, z)}{\partial x} \right|_{x=0.5, z=0.3}}{0.01} \\ &= \frac{\frac{y(0.5 + 0.02, 0.3) - y(0.5 + 0.01, 0.3)}{0.01} - \frac{y(0.5 + 0.01, 0.3) - y(0.5, 0.3)}{0.01}}{0.01} = 2.0000 \end{aligned} \quad (2.31)$$

$$\begin{aligned} \left. \frac{\partial^2 y(x, z)}{\partial x \partial z} \right|_{x=0.5, z=0.3} &= \frac{\partial}{\partial x} \left(\left. \frac{\partial y(x, z)}{\partial z} \right|_{x=0.5, z=0.3} \right) \approx \frac{\left. \frac{\partial y(x, z)}{\partial z} \right|_{x=0.5 + 0.01, z=0.3} - \left. \frac{\partial y(x, z)}{\partial z} \right|_{x=0.5, z=0.3}}{0.01} \\ &= \frac{\frac{y(0.5 + 0.01, 0.3 + 0.005) - y(0.5 + 0.01, 0.3)}{0.005} - \frac{y(0.5, 0.3 + 0.005) - y(0.5, 0.3)}{0.005}}{0.01} = -1.2106 \end{aligned} \quad (2.32)$$

Example 2.3

Simulate the following first-order differential equation using the Euler method with $\Delta t = 0.01$:

$$\frac{dy(t)}{dt} = -y(t) + u(t), \quad y(0) = 1 \quad (2.33)$$

$$u(t) = \begin{cases} 0 & t < 2.5 \\ 1 & t \geq 2.5 \end{cases} \quad (2.34)$$

Solution The MATLAB code in Table 2.4 is used to simulate the process in (2.33) and (2.34) and the simulation results are given in Figure 2.3.

2.2 Simulating Processes Including Time Delay

The historical data of a signal need to be stored to simulate the delayed signal. Consider the array variable $h_u(1 \times m)$ vector) that stores the historical data of $u(t)$ in the following form:

$h_u(1)$...	$h_u(m-3)$	$h_u(m-2)$	$h_u(m-1)$	$h_u(m)$
$u(t - (m-1)\Delta t)$...	$u(t - 3\Delta t)$	$u(t - 2\Delta t)$	$u(t - \Delta t)$	$u(t)$

where Δt is the sampling time. $h_u(m)$ should be the present value of $u(t)$ and $h_u(m-1)$ should be the delayed $u(t)$ by as much as one sampling time. That is, when a new present value

Table 2.4 MATLAB code to simulate Example 2.3.

<pre>euler_ex3.m clear; t=0.0; y=1.0; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; if (t<2.5) u=0; else u=1; end %Eq. (2.34) dy_dt=g_ex3(y,u); y=y+dy_dt*delta_t; %Eq. (2.11) t=t+delta_t; end figure(1); plot(t_array,y_array);</pre>	<pre>g_ex3.m function [dy_dt]=g_ex3(y,u) dy_dt=-y+u; end</pre>
	<pre>command window >> euler_ex3</pre>

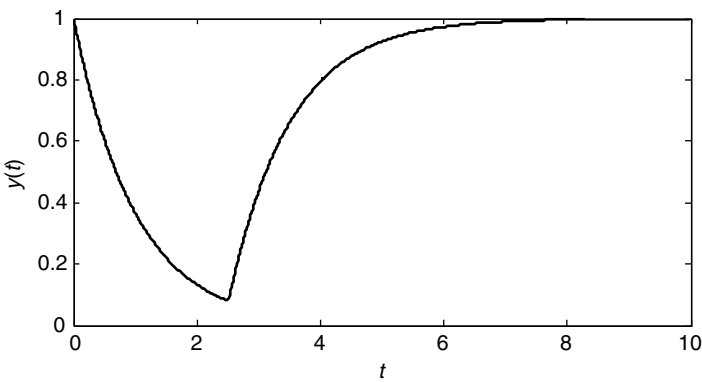


Figure 2.3 Simulation result of Example 2.3.

comes in, $h_u(j), j = 1, 2, \dots, m - 1$, should be replaced by $h_u(j + 1), j = 1, 2, \dots, m - 1$, and $h_u(m)$ should be replaced by the new value.

Now, assume that the time delay is θ , then $n\theta = \text{round}(\theta/\Delta t)$ is the number of the sampling time corresponding to θ . Where, $\text{round}(x)$ rounds x off to the nearest integer. Then, $h_u(m - n\theta)$ corresponds to the delayed signal $u(t - \theta)$.

For example, let us obtain $u(t - 1)$ for the signal

$$u(t) = 1 - \exp(-t), \quad t \geq 0 \tag{2.35}$$

$$u(t) = 0, \quad t < 0 \tag{2.36}$$

Table 2.5 MATLAB code to simulate the pure time delay.

delay_u.m	command window
<pre>clear; h_u=zeros(1,1000); theta=1.0; t_final=5.0; delta_t=0.01; t=0.0; n=round(t_final/delta_t); n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; %present time u=1-exp(-t); %present value for j = 1 : 999 h_u(j) = h_u(j+1) end h_u(1000) = u; %u(t) u_array(i) = u; %u(t) u_delay_array(i)=h_u(1000-n_theta); %delayed t=t+delta_t; end figure(1); plot(t_array,u_array,':',t_array,u_delay_array,'-'); legend('u(t)', 'u_{delay}(t)');</pre>	<pre>>> delay_u</pre>

The MATLAB code in Table 2.5 simulates the time delay and Figure 2.4 gives the simulation result.

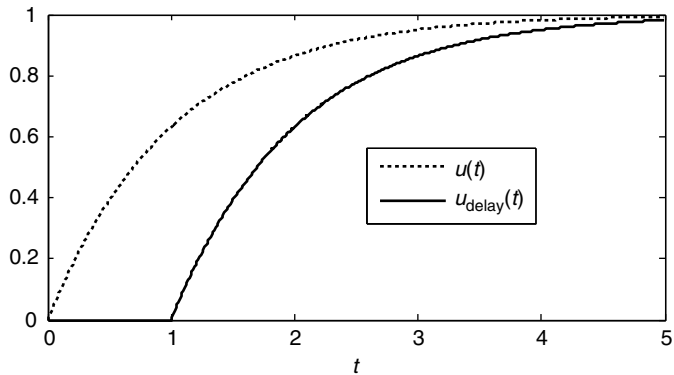


Figure 2.4 Simulation result of the pure time delay.

For the other example, consider the following high-order differential equation with time delay. This is the same as (2.16)–(2.18) except for the time delay.

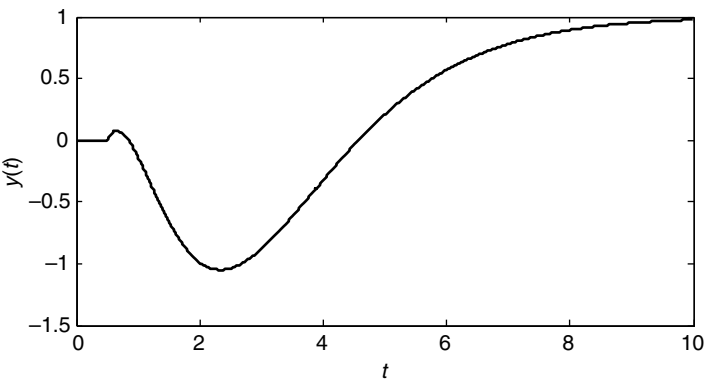


Figure 2.5 Simulation result of Table 2.6.

Table 2.6 MATLAB code to solve the high-order differential equation with time delay using the Euler method.

euler2_delay.m	g2.m
<pre>clear; t=0.0; x=[0 0 0]'; y=0; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.5; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; u=1.0; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; %u(t) dx_dt=g2(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,y_array);</pre>	<pre>function [dx_dt]=g2(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -5 1]'; dx_dt=A*x+B*u; end</pre>
	<pre>command window >> euler2_delay</pre>

$$\frac{d^3y(t)}{dt^3} + 3\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = \frac{d^2u(t-0.5)}{dt^2} - 5\frac{du(t-0.5)}{dt} + u(t-0.5) \quad (2.37)$$

$$u(t) = 1 \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (2.38)$$

$$\left. \frac{d^2y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (2.39)$$

The MATLAB code in Table 2.6 simulates (2.37)–(2.39) and Figure 2.5 gives the simulation result.

Example 2.4

Simulate the following first-order differential equation using the Euler method with $\Delta t = 0.01$:

$$\frac{dy(t)}{dt} = -y(t) + u(t-0.5), \quad y(0) = 1 \quad (2.40)$$

$$u(t) = 1 \quad \text{for } t \geq 2.5, \quad u(t) = 0 \quad \text{for } 0 \leq t < 2.5, \quad u(t) = 1 \quad \text{for } t < 0 \quad (2.41)$$

Solution In this example, the historical data $h_u(j) = 1, j = 1, 2, \dots, 1000$, should be initially filled with 1, because $u(t) = 1$ for $t < 0$. The MATLAB code to simulate the process (2.40)–(2.41) and the simulation results are given in Table 2.7 and Figure 2.6 respectively.

Example 2.5

Simulate the following second-order differential equation using the Euler method with $\Delta t = 0.01$:

$$2\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = -1.5\frac{du(t-0.2)}{dt} + 2.5u(t-0.2) \quad (2.42)$$

$$u(t) = 1 \quad \text{for } t \geq 3, \quad u(t) = 0.5 \quad \text{for } 0 \leq t < 3, \quad u(t) = 0.3 \quad \text{for } t < 0 \quad (2.43)$$

$$\left. \frac{dy(t)}{dt} \right|_{t=0} = -0.1, \quad y(0) = 0.8 \quad (2.44)$$

Solution In this example, the initial values are not zero. So, (1.184)–(1.186) should be used to determine the initial state $x(0)$. Then, the state-space realization is

$$\frac{dx(t)}{dt} = \begin{bmatrix} 0 & -1/2 \\ 1 & -3/2 \end{bmatrix} x(t) + \begin{bmatrix} 2.5/2 \\ -1.5/2 \end{bmatrix} u(t-0.2) \quad (2.45)$$

$$y(t) = \begin{bmatrix} 0 & 1 \end{bmatrix} x(t) \quad (2.46)$$

$$x(0) = \chi^{-1}\psi \quad (2.47)$$

Table 2.7 MATLAB code to simulate Example 2.4.

<pre>euler_delay_ex1.m clear; t=0.0; y=1.0; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); theta=0.5; % time delay h_u=ones(1,1000); %filled with 1 n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; if (t<2.5) u=0; else u=1; end for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dy_dt=g_delay_ex1(y,h_u(1000- n_theta)); y=y+dy_dt*delta_t; t=t+delta_t; end figure(1); plot(t_array,y_array);</pre>	<pre>g_delay_ex1.m function [dy_dt]=g_delay_ex1(y,u) dy_dt=-y+u; end</pre> <hr/> <pre>command window >> euler_delay_ex1</pre>
---	---

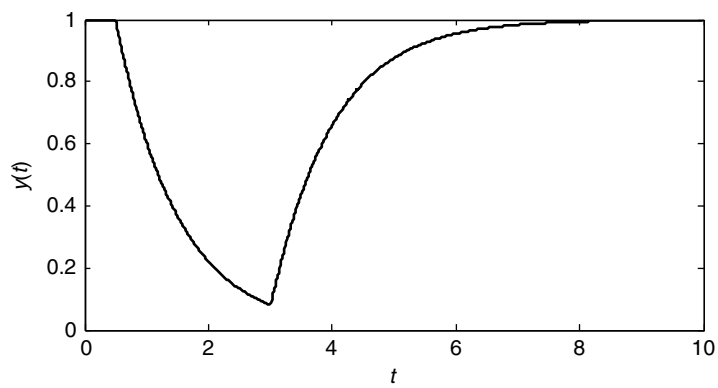


Figure 2.6 Simulation result of Example 2.4.

$$\psi = \begin{bmatrix} 0.8 \\ 0.125 \end{bmatrix} \quad (2.48)$$

$$\chi = \begin{bmatrix} 0 & 1 \\ 1 & -1.5 \end{bmatrix} \quad (2.49)$$

The historical data $h_u(j) = 0.3, j = 1, 2, \dots, 1000$, should be initially filled with 0.3, because $u(t) = 0.3$ for $t < 0$. The MATLAB code to simulate the process (2.42)–(2.44) and the simulation results are given in Table 2.8 and Figure 2.7 respectively.

Table 2.8 MATLAB code to simulate Example 2.5.

euler_delay_ex2.m	g_delay_ex2.m
<pre> clear; A=[0 -1/2; 1 -3/2]; B=[2.5/2; -1.5/2]; C=[0 1]; P=[0.8; -0.1-C*B*0.3]; %Eq. (2.49) K=[C; C*A]; %Eq. (2.48) x=inv(K)*P; %Eq. (2.47) t=0.0; y=0.8; t_final=10.0; delta_t=0.01; n=round(t_final/delta_t); theta=0.2; % time delay h_u=0.3*ones(1,1000); n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; if (t<3.0 & t>=0) u=0.5; end if (t>=3.0) u=1.0; end for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt= g_delay_ex2(y,x, h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,y_array); </pre>	<pre> function [dx_dt]= g_delay_ex2 (y,x,u) A=[0 -1/2; 1 -3/2]; B=[2.5/2; -1.5/2]; dx_dt=A*x+B*u; end </pre>
	<p>command window</p> <pre>>> euler_delay_ex2</pre>

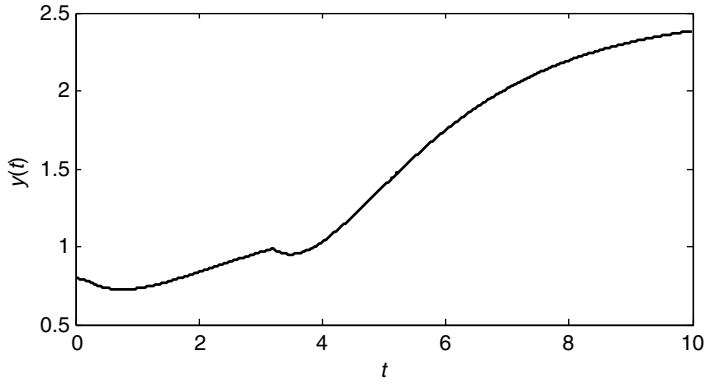


Figure 2.7 Simulation result of Example 2.5.

2.3 Simulating Closed-Loop Control Systems

If the process input is a function of the process output then it is a closed-loop control system. For example, consider the following control system:

$$2 \frac{dy(t)}{dt} = -y(t) + 5u(t - 0.3) \quad (2.50)$$

$$u(t) = 0.3(y_s(t) - y(t)) \quad (2.51)$$

where (2.50) is the process of which the input is $u(t)$ and output is $y(t)$. Equation (2.51) is a controller which determines the process input on the basis of the process output $y(t)$ and the external signal $y_s(t)$. So, the system (2.50) and (2.51) is a closed-loop control system. Previous simulation techniques can be extended to the closed-loop control system in a straightforward manner. Consider the following examples.

Example 2.6

Simulate the following third-order differential equation using the Euler method with $\Delta t = 0.01$:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = 0.1 \frac{d^2 u(t - 0.5)}{dt^2} - 0.8 \frac{du(t - 0.5)}{dt} + u(t - 0.5) \quad (2.52)$$

$$u(t) = 1.2(1 - y(t)) \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (2.53)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (2.54)$$

Solution In this example, the historical data $h_u(j) = 0, j = 1, 2, \dots, 1000$, should be initially filled with 0 because $u(t) = 0$ for $t < 0$. The MATLAB code to simulate the process (2.52)–(2.54) and the simulation results are given in Table 2.9 and Figure 2.8 respectively.

Table 2.9 MATLAB code to simulate the closed-loop control system of Example 2.6 using the Euler method.

euler_closed_ex1.m	g_closed_ex1.m
<pre> clear; t=0.0; x=[0 0 0]'; y=0; t_final=20.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.5; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; u=1.2*(1.0-y); %Eq. (2.53) u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_closed_ex1(y,x,h_u (1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,y_array); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_closed_ex1(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.8 0.1]'; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> euler_closed_ex1 </pre>

Example 2.7

Simulate the following third-order differential equation using the Euler method with $\Delta t = 0.01$:

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = -0.3 \frac{du(t-0.2)}{dt} + u(t-0.2) \quad (2.55)$$

$$u(t) = 1.5(y_s(t) - y(t)) + 1.5 \frac{d(y_s(t) - y(t))}{dt} \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (2.56)$$

$$y_s(t) = 1.0 \quad \text{for } t > 1, \quad y_s(t) = 0.0 \quad \text{for } t \leq 1 \quad (2.57)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (2.58)$$

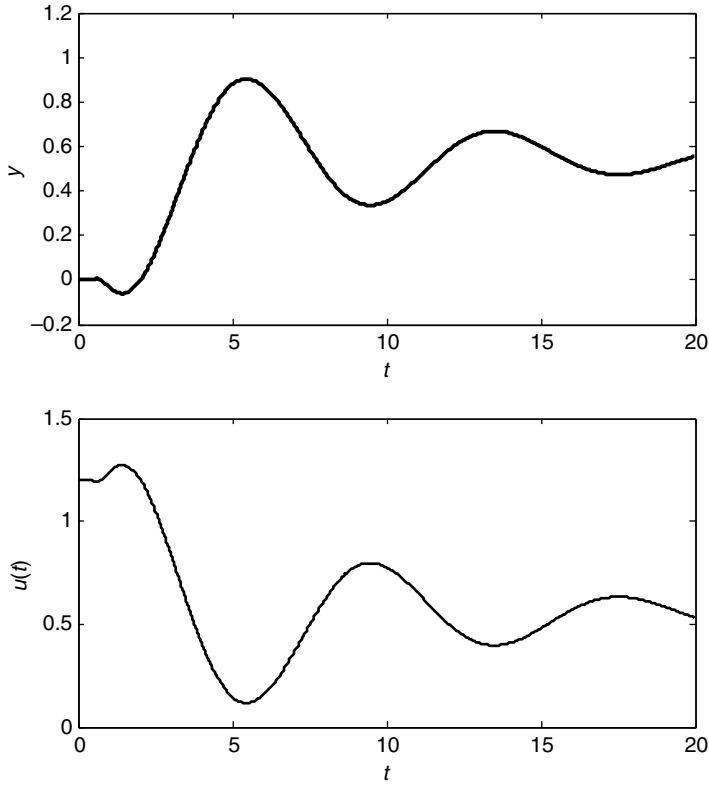


Figure 2.8 Simulation results in Example 2.6.

Solution In this example, the historical data $h_u(j) = 0, j = 1, 2, \dots, 1000$ should be initially filled with zero because $u(t) = 0$ for $t < 0$. The MATLAB code to simulate the process (2.55)–(2.58) and the simulation results are given in Table 2.10 and Figure 2.9 respectively. It uses the numerical derivative to implement $d(y_s(t) - y(t))/dt$.

2.4 Useful Numerical Analysis Methods

In this section, simple numerical analysis methods required to design the controller and simulate the dynamics of the control system are introduced.

2.4.1 Least-Squares Method

The least-squares method has been widely used to fit data to a linear function model. It estimates the model parameters (2.60) by minimizing the objective function of $Q(\hat{P})$.

$$\min_{\hat{P}} \left[Q(\hat{P}) = \sum_{i=1}^m (y_i - \hat{y}_i)^2 \right] \quad (2.59)$$

Table 2.10 MATLAB code to simulate the close-loop control system of Example 2.7 using the Euler method.

euler_closed_ex2.m	g_closed_ex2.m
<pre> clear; t=0.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; t_final=15.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); for i=1:n t_array(i)=t; y_array(i)=y; if (t>1) ys=1.0; else ys=0.0; end %Eq. (2.57) u=1.5*(ys-y)+1.5*((ys-y)-((ysb-yb))/delta_t; %Eq. (2.56) ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_closed_ex2(y,x,h_u (1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,y_array); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_closed_ex2(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> euler_closed_ex2 </pre>

$$\hat{y}_i = \hat{p}_1 \varphi_{1,i} + \hat{p}_2 \varphi_{2,i} + \hat{p}_3 \varphi_{3,i} + \cdots + \hat{p}_n \varphi_{n,i}, \quad i = 1, 2, 3, \dots, m \quad (2.60)$$

where $\hat{p}_1, \hat{p}_2, \hat{p}_3, \dots, \hat{p}_n$ are the model parameters. y_i and $\varphi_{k,i}$ are the given data. Equations (2.59) and (2.60) can be rewritten as (2.61) and (2.62):

$$\min_{\hat{P}} [Q(\hat{P}) = (Y - \hat{Y})^T (Y - \hat{Y})] \quad (2.61)$$

$$\hat{Y} = \Phi \hat{P} \quad (2.62)$$

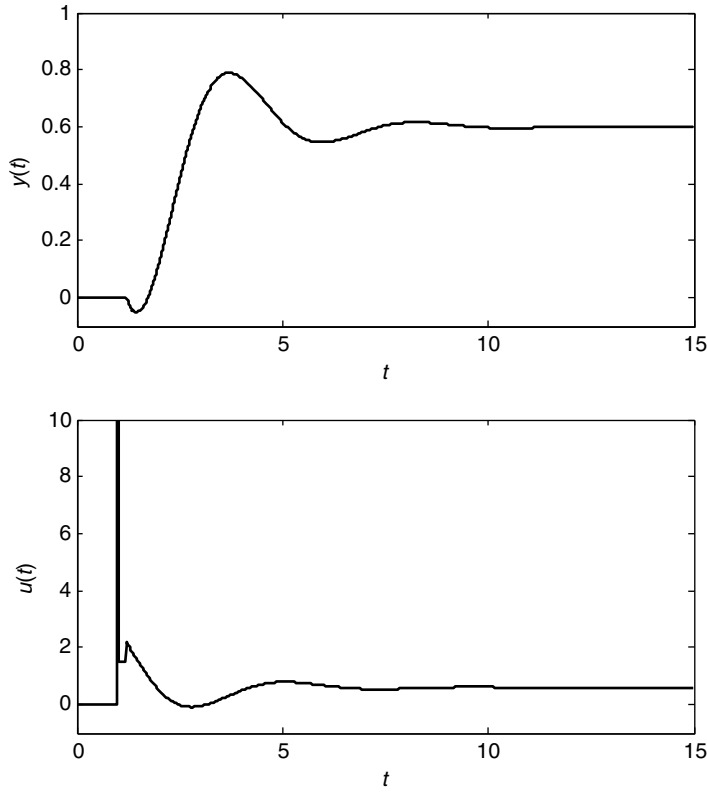


Figure 2.9 Simulation results in Example 2.7.

where the matrices and vectors of (2.61) and (2.62) are

$$Y = \begin{bmatrix} y_1 \\ y_2 \\ \vdots \\ y_m \end{bmatrix}, \quad \hat{Y} = \begin{bmatrix} \hat{y}_1 \\ \hat{y}_2 \\ \vdots \\ \hat{y}_m \end{bmatrix}, \quad \Phi = \begin{bmatrix} \varphi_{1,1} & \varphi_{2,1} & \cdots & \varphi_{n,1} \\ \varphi_{1,2} & \varphi_{2,2} & \cdots & \varphi_{n,2} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{1,m} & \varphi_{2,m} & \cdots & \varphi_{n,m} \end{bmatrix}, \quad \hat{P} = \begin{bmatrix} \hat{p}_1 \\ \hat{p}_2 \\ \vdots \\ \hat{p}_n \end{bmatrix} \quad (2.63)$$

Y , \hat{Y} , Φ and \hat{P} are an $m \times 1$ vector, an $m \times 1$ vector, an $m \times n$ matrix and an $n \times 1$ vector respectively. Note that the solution of the optimization problem (2.61) can be obtained by finding \hat{P} that makes the gradient of $Q(\hat{P})$ zero. So, consider the following gradient of $Q(\hat{P})$:

$$\frac{\partial Q(\hat{P})}{\partial \hat{P}} = -2 \left(\frac{\partial \hat{Y}}{\partial \hat{P}} \right)^T (Y - \hat{Y}) = -2 \Phi^T (Y - \Phi \hat{P}) \quad (2.64)$$

where the definitions of the gradients in (2.64) are

$$\frac{\partial Q(\hat{P})}{\partial \hat{P}} = \left[\frac{\partial Q(\hat{P})}{\partial \hat{p}_1} \quad \frac{\partial Q(\hat{P})}{\partial \hat{p}_2} \quad \dots \quad \frac{\partial Q(\hat{P})}{\partial \hat{p}_n} \right]^T \quad (2.65)$$

$$\frac{\partial \hat{Y}}{\partial \hat{P}} = \begin{bmatrix} \frac{\partial \hat{y}_1}{\partial \hat{p}_1} & \frac{\partial \hat{y}_1}{\partial \hat{p}_2} & \dots & \frac{\partial \hat{y}_1}{\partial \hat{p}_n} \\ \frac{\partial \hat{y}_2}{\partial \hat{p}_1} & \frac{\partial \hat{y}_2}{\partial \hat{p}_2} & \dots & \frac{\partial \hat{y}_2}{\partial \hat{p}_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial \hat{y}_m}{\partial \hat{p}_1} & \frac{\partial \hat{y}_m}{\partial \hat{p}_2} & \dots & \frac{\partial \hat{y}_m}{\partial \hat{p}_n} \end{bmatrix} \quad (2.66)$$

From (2.64), it is clear that the solution of the optimization problem (2.61) should satisfy (2.67).

$$\Phi^T(Y - \Phi \hat{P}) = \Phi^T Y - \Phi^T \Phi \hat{P} = 0 \quad (2.67)$$

From (2.67), obtain the optimal solution:

$$\hat{P} = [\Phi^T \Phi]^{-1} [\Phi^T Y] \quad (2.68)$$

It is remarkable that the estimate (2.68) minimizes the sum of the squares' errors (2.59). So, (2.68) is called the least-squares method.

Example 2.8

Obtain the model of which the structure is given in (2.69) to fit the (x, y) data of (1.1, 3.1), (1.9, 5.1), (3.1, 6.9), (4.0, 9.2) in the sense of least squares.

$$\hat{y} = \hat{p}_1 + \hat{p}_2 x \quad (2.69)$$

Solution From the data:

$$Y = \begin{bmatrix} 3.1 \\ 5.1 \\ 6.9 \\ 9.2 \end{bmatrix}, \quad \Phi = \begin{bmatrix} 1 & 1.1 \\ 1 & 1.9 \\ 1 & 3.1 \\ 1 & 4.0 \end{bmatrix} \quad (2.70)$$

Then, the least-squares method (2.68) results in the following estimates:

$$\hat{P} = [\Phi^T \Phi]^{-1} [\Phi^T Y] = \begin{bmatrix} 0.9853 \\ 2.0157 \end{bmatrix} \quad (2.71)$$

That is, $\hat{p}_1 = 0.9853$ and $\hat{p}_2 = 2.0157$. The estimated model output (2.62) for the given x and the estimated \hat{P} is

$$\hat{Y} = \Phi \hat{P} = \begin{bmatrix} 3.2026 \\ 4.8152 \\ 7.2340 \\ 9.0482 \end{bmatrix} \quad (2.72)$$

Example 2.9

Obtain the model parameters of τ and ξ in

$$\tau^4 |G(i\omega_k)|^2 \omega_k^4 + (4\tau^2 \xi^2 - 2\tau^2) |G(i\omega_k)|^2 \omega_k^2 = 1 - |G(i\omega_k)|^2, \quad \omega_k = 0.1k, k = 1, 2, 3, \dots, 10 \quad (2.73)$$

$$G(s) = \frac{1}{(s+1)^3} \quad (2.74)$$

Solution Let us define $\hat{p}_1 = \tau^4$, $\varphi_{1,k} = |G(i\omega_k)|^2 \omega_k^4$, $\hat{p}_2 = 4\tau^2 \xi^2 - 2\tau^2$, $\varphi_{2,k} = |G(i\omega_k)|^2 \omega_k^2$ and $y_k = 1 - |G(i\omega_k)|^2$. Then, $\tau = \hat{p}_1^{1/4}$ and $\xi = \sqrt{(\hat{p}_2 + 2\tau^2)/(4\tau^2)}$. The source code for the least-squares method is shown in Table 2.11. The estimates are $\tau = 1.424$ and $\xi = 0.917$. The model output and the real data are compared in Figure 2.10. This is one of the model reduction methods, which will be discussed later in this book.

Table 2.11 MATLAB code to solve Example 2.9 using the least-squares method.

ls_ex2.m	g_ls_ex2.m
<pre>clear; for k=1:10 w(k)=k*0.1; G(k)=g_ls_ex2(w(k)); y(k,1)=1-(abs(G(k))^2); phi_1(k,1)=(abs(G(k))^2)*w(k)^4; phi_2(k,1)=(abs(G(k))^2)*w(k)^2; end PHI=[phi_1 phi_2]; Y=y; P_hat=inv(PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/(4*tau^2)) ^0.5; Y_hat=PHI*P_hat; fprintf('P_hat = %5.3f %5.3f \n', P_hat(1), P_hat(2)); fprintf('tau = %5.3f xi = %5.3f \n', tau, xi); figure(1); plot(1:10, Y, 1:10, Y_hat); legend('real data', 'model output');</pre>	<pre>function [G]=g_ls_ex2(w) s=i*w; G=1/(s+1)^3; end</pre>
	<p>command window</p> <pre>>> ls_ex2 P_hat = 4.116 2.764 tau = 1.424 xi = 0.917</pre>

2.4.2 Root-Finding Methods

A root-finding method finds the x value that satisfies $f(x) = 0$. This section introduces the bisection method, followed by the Newton–Raphson method.

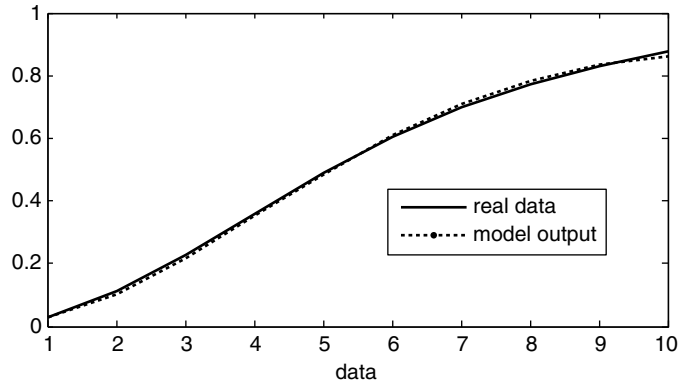


Figure 2.10 Estimation results of the least-squares method in Example 2.9.

The bisection method finds the root by reducing the interval (in which the root exists) by half at each iteration. Consider Figure 2.11 to understand the principle of the bisection method. It is assumed that the initial interval is chosen to include the root. Let the left-hand-side value of the interval be x_L and the right-hand-side value of the interval be x_R and the interval is $x_R - x_L$. In the first iteration, the bisection method divides the initial interval by 2 and calculates the middle point $x_M = (x_R + x_L)/2$. If the sign of $f(x_R)$ and the sign of $f(x_M)$ are the same, then it reduces the interval to half by updating the right-hand-side value as $x_R = x_M$. If the sign of $f(x_R)$ and the sign of $f(x_M)$ are not the same, then it updates the left-hand-side value as $x_L = x_M$. In the second iteration, the bisection method repeats the same procedure with the new interval. It repeats the procedure and updates the interval until the interval becomes a sufficiently small value.

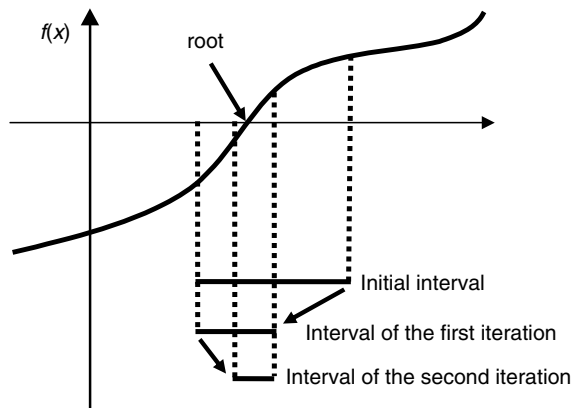


Figure 2.11 Graphical representation of the bisection method.

Example 2.10

Obtain the root of $f(x) = \exp(x) + 3x - 20$ with the initial interval between $x = -1$ and $x = 5$.

Solution The source code for the bisection method and the estimated root ($x = 2.52074$) are shown in Table 2.12.

Table 2.12 MATLAB code for the bisection method in Example 2.10.

bs_ex1.m	g_bs_ex1.m
<pre>clear; x_L = -1; x_R = 5; while (1) x_M=(x_L+x_R)/2.0; f_M=g_bs_ex1(x_M); f_R=g_bs_ex1(x_R); if (f_M*f_R > 0) x_R=x_M; else x_L=x_M; end if (abs(x_R-x_L)<0.000000001) break; end end fprintf('x = %7.5f f = %7.5f\n', x_L, f_R);</pre>	<pre>function [f]=g_bs_ex1(x) f = exp(x)+3*x-20.0; end</pre>
	<div>command window</div> <pre>>> bs_ex1 x = 2.52074 f = 0.00000</pre>

The Newton–Raphson method finds the root of $f(x) = 0$ by approximating the given function with a linear function and calculating the root of the linear function in a repetitive manner. Let us approximate the given nonlinear function $f(x)$ with the first-order Taylor series expansion at the k th iteration ($x = x_k$) as below:

$$f(x) \approx f(x_k) + \left. \frac{df}{dx} \right|_{x=x_k} (x - x_k)$$

(2.75)

Then, the approximated root ($x = x_{k+1}$) of the next iteration can be obtained by setting $f(x) = 0$ in (2.75) as shown in (2.76):

$$x_{k+1} = x_k - \frac{f(x_k)}{\left. df/dx \right|_{x=x_k}}$$

(2.76)

The Newton–Raphson method finds the root by repeating (2.76) until the distance between x_{k+1} and x_k or the absolute value of $f(x_{k+1})$ becomes a sufficiently small value.

As mentioned above, the Newton–Raphson method uses the approximated linear function to calculate the next approximated root. It should be noted that the approximation using the linear function results in serious linearization errors if the original function of $f(x)$ is highly nonlinear. Then, the performance of the Newton–Raphson method will be degraded significantly,

resulting in a poor convergence rate or divergence. To incorporate such a case, the following modification can be used:

$$x_{\text{new}} = x_k - \frac{f(x_k)}{df/dx|_{x=x_k} + \beta} \quad (2.77)$$

$$\text{If } |f(x_{\text{new}})| < |f(x_k)|, \text{ set } \beta = \beta/1.5, x_{k+1} = x_{\text{new}} \text{ and perform the next iteration} \quad (2.78)$$

$$\text{If } |f(x_{\text{new}})| \geq |f(x_k)|, \text{ set } \beta = 1.5\beta, x_{k+1} = x_k \text{ (no update) and perform the next iteration} \quad (2.79)$$

where the initial β value is a small positive value. $|f(x_k)|$ denotes the absolute value of $f(x_k)$. This rule of thumb updates x only if the function value decreases; that is, $|f(x_{\text{new}})| < |f(x_k)|$.

The above-mentioned derivations can be extended to multivariable and multifunction systems. Consider the following root-finding system:

$$\begin{aligned} f_1(x_1, x_2, \dots, x_n) &= 0 \\ f_2(x_1, x_2, \dots, x_n) &= 0 \\ &\vdots \\ f_n(x_1, x_2, \dots, x_n) &= 0 \end{aligned} \quad (2.80)$$

Let us introduce the following vector representation to describe (2.80) more simply.

$$\mathbf{F}(\mathbf{X}) = 0 \quad (2.81)$$

where $\mathbf{X} = [x_1 \ x_2 \ \dots \ x_n]^T$ and $\mathbf{F} = [f_1 \ f_2 \ \dots \ f_n]^T$. Equation (2.81) can be linearized by the first-order Taylor series expansion as follows:

$$\mathbf{F}(\mathbf{X}) \approx \mathbf{F}(\mathbf{X}_k) + \nabla \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k} (\mathbf{X} - \mathbf{X}_k) \quad (2.82)$$

where $\nabla \mathbf{F}$ is the gradient of the nonlinear function (2.80) or (2.81).

$$\nabla \mathbf{F} = \begin{bmatrix} \frac{\partial f_1}{\partial x_1} & \frac{\partial f_1}{\partial x_2} & \dots & \frac{\partial f_1}{\partial x_n} \\ \frac{\partial f_2}{\partial x_1} & \frac{\partial f_2}{\partial x_2} & \dots & \frac{\partial f_2}{\partial x_n} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \frac{\partial f_n}{\partial x_2} & \dots & \frac{\partial f_n}{\partial x_n} \end{bmatrix} \quad (2.83)$$

Now, the approximated roots of the next iteration (2.84) can be obtained by setting $\mathbf{F}(\mathbf{X}) = 0$ in (2.82).

$$\mathbf{X}_{k+1} = \mathbf{X}_k - (\nabla \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k})^{-1} \mathbf{F}(\mathbf{X}_k) \quad (2.84)$$

The Newton–Raphson method for the multivariable and multifunction system repeats (2.84) until the norm of $\mathbf{X}_{k+1} - \mathbf{X}_k$ or the norm of $\mathbf{F}(\mathbf{X}_k)$ becomes a sufficiently small value. This

method also shows a poor convergence rate or divergence for a highly nonlinear function system. To incorporate the case, the following modification can be used:

$$\mathbf{X}_{\text{new}} = \mathbf{X}_k - (\nabla \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k} + \beta \mathbf{I})^{-1} \mathbf{F}(\mathbf{X}_k) \tag{2.85}$$

If $\|\mathbf{F}(\mathbf{X}_{\text{new}})\| < \|\mathbf{F}(\mathbf{X}_k)\|$, set $\beta = \beta/1.5$, $\mathbf{X}_{k+1} = \mathbf{X}_{\text{new}}$ and perform the next iteration (2.86)

If $\|\mathbf{F}(\mathbf{X}_{\text{new}})\| \geq \|\mathbf{F}(\mathbf{X}_k)\|$, set $\beta = 1.5\beta$, $\mathbf{X}_{k+1} = \mathbf{X}_k$ (no update) and perform the next iteration (2.87)

where the initial β value is a small positive value and \mathbf{I} is the $n \times n$ unit matrix. $\|\mathbf{F}(\mathbf{X}_k)\|$ denotes the Euclidean norm of $\mathbf{F}(\mathbf{X}_k)$. This rule of thumb updates \mathbf{X} only if the function value decreases (that is, $\|\mathbf{F}(\mathbf{X}_{\text{new}})\| < \|\mathbf{F}(\mathbf{X}_k)\|$).

Example 2.11

Obtain the root of $f(x) = \exp(x) + 3x - 20$ with the initial value of $x_0 = -1.0$.

Solution The source code for the Newton–Raphson method and the estimated root ($x = 2.52074$) are shown in Table 2.13.

Table 2.13 MATLAB code for the Newton–Raphson method in Example 2.11.

<pre>nr_ex1.m clear; x = -1; beta = 0.001; while (1) f = g_nr_ex1(x); df = deri_nr_ex1(x); x_new = x-f/(df+beta); f_new = g_nr_ex1(x_new); if(abs(f_new) < abs(f)) beta = beta/1.5; x = x_new; else beta = beta*1.5; end if(abs(f_new)<0.000000001) break; end end fprintf(']>'x = %7.5f f = %7.5f \n', x, f_new);</pre>	<pre>g_nr_ex1.m function [f]=g_nr_ex1(x) f = exp(x)+3*x-20.0; end deri_nr_ex1.m function [df]=deri_nr_ex1(x) df = exp(x)+3; end command window >> nr_ex1 x = 2.52074 f = 0.00000</pre>
--	---

Example 2.12

Obtain the roots of $f_1(x_1, x_2) = x_1^2 + x_2 + \exp(2x_2) - 2 = 0$ and $f_2(x_1, x_2) = x_1x_2^3 + \exp(x_1 - 1) - 1 = 0$ with the initial values of $x_{1,0} = 0.5$ and $x_{2,0} = 0.5$.

Solution The source code for the Newton–Raphson method and the estimated roots ($x_1 = 1.0$ and $x_2 = 0.0$) are shown in Table 2.14.

Table 2.14 MATLAB code for the Newton–Raphson method in Example 2.12.

<pre>nr_ex2.m clear; x = [0.5; 0.5]; beta = 0.001; while (1) f = g_nr_ex2(x); df = deri_nr_ex2(x); x_new = x-inv(df +beta*eye(2,2))*f; f_new = g_nr_ex2(x_new); if(abs(f_new) < abs(f)) beta = beta/1.5; x = x_new; else beta = beta*1.5; end if(abs(f_new)<0.000001) break; end end fprintf('x(1) = %7.5f x(2) = %7.5f\n',x(1),x(2)); fprintf('f(1) = %7.5f f(2) = %7.5f\n',f_new(1),f(2));</pre>	<pre>g_nr_ex2.m function [f]=g_nr_ex2(x) f(1,1) = x(1)^2+x(2)+exp(2*x(2))-2; f(2,1) = x(1)*x(2)^3+exp(x(1)-1)-1; end deri_nr_ex2.m function [df]=deri_nr_ex2(x) df(1,1) = 2*x(1); df(1,2) = 1+2*exp(2*x(2)); df(2,1) = x(2)^3+exp(x(1)-1); df(2,2) = 3*x(1)*x(2)^2; end command window >> nr_ex2 x(1) = 1.00000 x(2) = 0.00000 f(1) = 0.00000 f(2) = 0.00000</pre>
---	---

Example 2.13

Obtain the root of $f(x) = \exp(x) + 3x - 20$ with the initial value of $x_0 = -1.0$. Use a numerical derivative to calculate the derivative in the Newton–Raphson method.

Solution The source code for the Newton–Raphson method with the numerical derivative and the estimated root ($x = 2.52074$) are shown in Table 2.15.

2.4.3 Numerical Integration

The numerical integration method has been widely used to implement the process controller and simulate the dynamic behavior of the closed-loop control system. Two simple methods will be presented in this section.

The simplest one is the Euler integration method, which is sufficient for most cases in the research area of the process control and identification. Consider the differential equation $ds(t)/dt = e(t)$. Then, $s(t)$ is the integral of $e(t)$. From the Euler approximation $ds(t)/dt \approx (s(t + \Delta t) - s(t))/\Delta t$, the numerical integration $s(t + \Delta t) = s(t) + e(t)\Delta t$ is obtained. The integral with respect to t can be calculated by repeating $s(t + \Delta t) = s(t) + e(t)\Delta t$.

On the other hand, the other Euler approximation $ds(t)/dt \approx (s(t) - s(t - \Delta t))/\Delta t$ (that is, $s(t) = s(t - \Delta t) + e(t)\Delta t$) is also possible. Then, the numerical integration $s(t + \Delta t) = s(t) + e(t + \Delta t)\Delta t$ is obtained. The trapezoidal integration method $s(t + \Delta t) = s(t) + (e(t + \Delta t) +$

Table 2.15 MATLAB code for the Newton–Raphson method in Example 2.13.

<div>nr_ex3.m</div> <pre>clear; x = -1; beta = 0.001; while (1) f = g_nr_ex3(x); df = deri_nr_ex3(x); x_new = x-f/(df+beta); f_new=g_nr_ex3(x_new); if(abs(f_new) < abs(f)) beta = beta/1.5; x = x_new; else beta = beta*1.5; end if(abs(f_new) <0.000000001) break; end end fprintf('x=%7.5f f=%7.5f \n',x,f_new);</pre>	<div>g_nr_ex3.m</div> <pre>function [f]=g_nr_ex3(x) f = exp(x)+3*x-20.0; end</pre>
	<div>deri_nr_ex3.m</div> <pre>function [df]=deri_nr_ex3(x) f_delta = g_nr_ex3(x+0.0001); f = g_nr_ex3(x); df = (f_delta - f)/0.0001; end</pre>
	<div>command window</div> <pre>>> nr_ex3 x = 2.52074 f = 0.00000</pre>

$e(t))\Delta t/2$ is obtained by averaging the two Euler approximations $s(t + \Delta t) = s(t) + e(t)\Delta t$ and $s(t + \Delta t) = s(t) + e(t + \Delta t)\Delta t$. The trapezoidal method is usually superior to the Euler method. But, the difference of the performances is negligible for most cases if a sufficiently small Δt is used.

Example 2.14

Obtain the numerical integration of $f(x) = \exp(x) + 3x - 20$ from $x = 0$ to $x = 1$ using the Euler method and the trapezoidal method.

Solution The source code for the Euler method and the trapezoidal method and the results are shown in Table 2.16. The analytical value is $-16.781\,72$.

Example 2.15

Obtain $s(t) = s(0) + \int_0^t e(\tau) \, d\tau$ for $e(t) = t \exp(-t)$ and $s(0) = 1.5$ using the Euler method.

Solution The source code for the Euler method and the results are shown in Table 2.17. and Figure 2.12 respectively.

2.4.4 Optimization Methods

Simple optimization methods, such as the interval halving method and the Levenberg–Marquardt method, have been used in the research area of process identification. The interval halving method for a single-variable optimization is introduced in this section, followed by the

Table 2.16 MATLAB code for the numerical integration method in Example 2.14.

<pre>ni_ex1.m clear; se = 0.0; st = 0.0; x = 0.0; n = 1000; delta_x = (1.0-0.0)/n; for i = 1:n f1 = g_ni_ex1(x); se = se + f1*delta_x; x = x + delta_x; f2 = g_ni_ex1(x); st = st + (f1+f2) *delta_x/2; end fprintf('Euler = %7.5f\n',se); fprintf('Trapezoidal = %7.5f\n',st);</pre>	<pre>g_ni_ex1.m function [f]=g_ni_ex1(x) f = exp(x)+3*x-20.0; end</pre> <hr/> <pre>command window >> ni_ex1 Euler = -16.78408 Trapezoidal = -16.78172</pre>
--	---

Table 2.17 MATLAB code for the numerical integration method in Example 2.15.

<pre>ni_ex2.m clear; se = 1.5; % initial value t = 0.0; n = 1000; tf = 5.0; delta_t = (tf-0.0)/n; for i = 1:n t_array(i)=t; se_array(i)=se; f = g_ni_ex2(t); se = se + f*delta_t; t = t + delta_t; end figure(1); plot(t_array,se_array);</pre>	<pre>g_ni_ex2.m function [f]=g_ni_ex2(t) f = t*exp(-t); end</pre> <hr/> <pre>command window >> ni_ex2</pre>
--	---

Levenberg–Marquardt method to solve multivariable optimization method. The optimization methods are to find the optimal solution that minimizes the objective function as $\min_x f(x)$.

The interval halving method removes exactly half of the interval at each iteration. Consider the three cases $f(x_1) < f(x_2) \leq f(x_3)$, $f(x_1) \geq f(x_2) > f(x_3)$ and $f(x_1) \geq f(x_2)$ and $f(x_2) \leq f(x_3)$ shown in Figure 2.13. x_L , x_1 , x_2 , x_3 and x_R are equally spaced. The optimal solution may exist

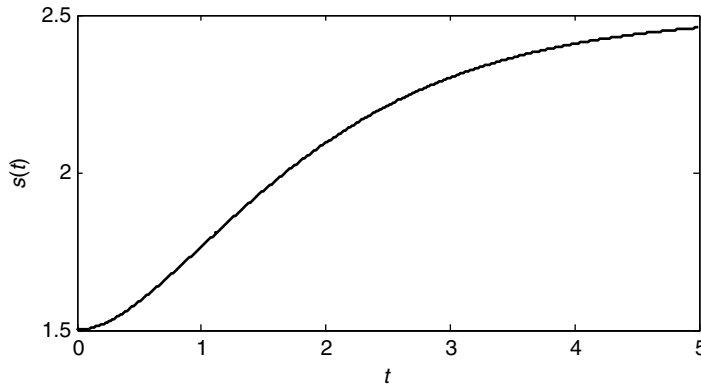


Figure 2.12 The result of the numerical integration in Example 2.15.

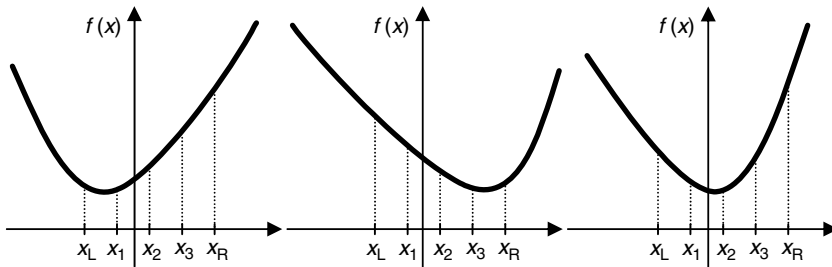


Figure 2.13 Three cases for the location of the optimal solution.

between x_L and x_2 , x_2 and x_R , x_1 and x_3 for each case of $f(x_1) < f(x_2) \leq f(x_3)$, $f(x_1) \geq f(x_2) > f(x_3)$, $f(x_1) \geq f(x_2)$ and $f(x_2) \leq f(x_3)$ respectively. So, the subintervals (x_L, x_1) and (x_3, x_R) can be removed for the third case. Similarly, the subintervals (x_2, x_R) and (x_L, x_2) can be eliminated for the first case and the second case respectively. Then, the procedure of the interval halving method will have the following three steps. Step 1, calculate x_1 , x_2 and x_3 for given the interval (x_L, x_R) and obtain $f(x_1)$, $f(x_2)$ and $f(x_3)$. Step 2, set $x_R = x_2$ and $x_L = x_2$ for the first case and the second case respectively. And, set $x_R = x_3$ and $x_L = x_1$ for the third case. Step 3, finish if the interval is sufficiently small. Otherwise, repeat the procedure from Step 1 to Step 3.

Example 2.16

Obtain the optimal solution for the following optimization problem with the initial interval from -5 to 3 :

$$\min_x f(x) \text{ subject to } f(x) = (x-1)^2 \quad (2.88)$$

Solution The source code for the interval halving method and the results are shown in Table 2.18.

Table 2.18 MATLAB code for the interval halving method in Example 2.16.

ih_ex1.m	g_ih_ex1.m
<pre> clear; x_L = -5.0; x_R = 3.0; while (1) delta_x = (x_R - x_L) / 4; x1 = x_L + delta_x; x2 = x_L + 2 * delta_x; x3 = x_L + 3 * delta_x; f1 = g_ih_ex1(x1); f2 = g_ih_ex1(x2); f3 = g_ih_ex1(x3); if ((f1 < f2) & (f2 <= f3)) x_R = x2; end if ((f1 >= f2) & (f2 > f3)) x_L = x2; end if ((f1 >= f2) & (f2 <= f3)) x_L = x1; x_R = x3; end if (abs(x_R - x_L) < 0.00000001) break; end end fprintf('x = %7.5f\n', x2); fprintf('f = %7.5f\n', f2); </pre>	<pre> function [f]=g_ih_ex1(x) f = (x-1)^2; end </pre> <hr/> <p style="text-align: center;">command window</p> <pre> >> ih_ex1 x = 1.00000 f = 0.00000 </pre>

Note that the gradient of the objective function is zero at the optimal solution. On the basis of this fact, the Levenberg–Marquardt method estimates the optimal solution by finding the roots that make the gradient of the objective function zero.

$$\nabla \mathbf{F}(\mathbf{X}) = 0 \quad (2.89)$$

where

$$\nabla \mathbf{F} = \begin{bmatrix} \frac{\partial f}{\partial x_1} & \frac{\partial f}{\partial x_2} & \cdots & \frac{\partial f}{\partial x_n} \end{bmatrix}^T$$

is the gradient of the objective function $f(x_1, x_2, \dots, x_n)$ and $\mathbf{X} = [x_1 x_2 \cdots x_n]^T$. $\nabla \mathbf{F}(\mathbf{X})$ can be approximated by the first-order Taylor series expansion at the k th iteration as follows:

$$\nabla \mathbf{F}(\mathbf{X}) \approx \nabla \mathbf{F}(\mathbf{X}_k) + \nabla^2 \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k} (\mathbf{X} - \mathbf{X}_k) \quad (2.90)$$

where $\nabla^2 \mathbf{F}$ is the Jacobian of the objective function $f(x_1, x_2, \dots, x_n)$.

$$\nabla^2 \mathbf{F} = \begin{bmatrix} \frac{\partial^2 f}{\partial x_1 \partial x_1} & \frac{\partial^2 f}{\partial x_2 \partial x_1} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_1} \\ \frac{\partial^2 f}{\partial x_1 \partial x_2} & \frac{\partial^2 f}{\partial x_2 \partial x_2} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_2} \\ \vdots & \vdots & \ddots & \vdots \\ \frac{\partial^2 f}{\partial x_1 \partial x_n} & \frac{\partial^2 f}{\partial x_2 \partial x_n} & \cdots & \frac{\partial^2 f}{\partial x_n \partial x_n} \end{bmatrix} \quad (2.91)$$

Now the approximated roots of the next iteration in (2.92) can be obtained by setting $\nabla \mathbf{F}(\mathbf{X}) = 0$ in (2.90).

$$\mathbf{X}_{k+1} = \mathbf{X}_k - (\nabla^2 \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k})^{-1} \nabla \mathbf{F}(\mathbf{X}_k) \quad (2.92)$$

The optimal solution can be obtained by repeating (2.92) until the norm of $\mathbf{X}_{k+1} - \mathbf{X}_k$ or the norm of $\nabla \mathbf{F}(\mathbf{X}_k)$ becomes a sufficiently small value. But this approach would show a poor convergence rate or divergence for a highly nonlinear function system. To incorporate such a case, the following Levenberg–Marquardt method is introduced:

$$\mathbf{X}_{\text{new}} = \mathbf{X}_k - (\nabla^2 \mathbf{F}|_{\mathbf{X}=\mathbf{X}_k} + \beta \mathbf{I})^{-1} \nabla \mathbf{F}(\mathbf{X}_k) \quad (2.93)$$

$$\text{If } f(\mathbf{X}_{\text{new}}) < f(\mathbf{X}_k), \text{ set } \beta = \beta/1.5, \mathbf{X}_{k+1} = \mathbf{X}_{\text{new}} \text{ and perform the next iteration} \quad (2.94)$$

$$\text{If } f(\mathbf{X}_{\text{new}}) \geq f(\mathbf{X}_k), \text{ set } \beta = 1.5\beta, \mathbf{X}_{k+1} = \mathbf{X}_k \text{ (no update) and perform the next iteration} \quad (2.95)$$

where the initial β value is a small positive value and \mathbf{I} is the $n \times n$ unit matrix. It updates \mathbf{X} only if the function value decreases (that is, $f(\mathbf{X}_{\text{new}}) < f(\mathbf{X}_k)$).

Example 2.17

Obtain the optimal solution for the following optimization problem with the initial values $x_1 = 2.5$, $x_2 = 0.0$ and $x_3 = 0.0$:

$$\min_{x_1, x_2, x_3} f(x_1, x_2, x_3) \text{ subject to } f(x) = (x_1 - 1)^4 (x_2 - 2)^2 + (x_3 - 3)^2 \quad (2.96)$$

Solution The source code for the Levenberg–Marquardt method and the results are shown in Table 2.19.

Table 2.19 MATLAB code for the Levenberg–Marquardt method in Example 2.17.

lm_ex1.m	g_lm_ex1.m
<pre>clear; x = [2.5; 0.0; 0.0]; beta = 0.001; while (1) f = g_lm_ex1(x); g = gradi_lm_ex1(x); c = conj_lm_ex1(x); x_new = x - inv(c + beta * eye(3, 3)) * g; f_new = g_lm_ex1(x_new); if (abs(f_new) < abs(f)) beta = beta/1.5; x = x_new; else beta = beta*1.5; end end</pre>	<pre>function [f]=g_lm_ex1(x) f = (x(1)-1)^4*(x(2)-2)^2+(x(3)-3)^2; end</pre>
	<pre>gradi_lm_ex1.m function [g]=gradi_lm_ex1(x) g = [4*(x(1)-1)^3*(x(2)-2)^2 (x(1)-1)^4*2*(x(2)-2) 2*(x(3)-3)]; end</pre>

Table 2.19 (Continued)

<pre> if(abs(g)<0.00000001) break; end end fprintf('x(1)=%7.5f, x(2)=%7.5f, x(3)=%7.5f\n',x(1),x(2),x(3)); fprintf('f = %7.5f\n',f); </pre>	<pre> command window >> lm_ex1 x(1)=1.01560, x(2)=1.98952, x(3)=3.00000 f = 0.00000 </pre>
<pre> conj_lm_ex1.m function [c]=conj_lm_ex1(x) c= [12*(x(1)-1)^2*(x(2)-2)^2 8*(x(1)-1)^3*(x(2)-2) 0 8*(x(1)-1)^3*(x(2)-2) (x(1)-1)^4*2 0 0 0 2]; end </pre>	

Problems

2.1 Calculate the numerical derivative $dy(t)/dt$ at $t = 0.3$ with $\Delta t = 0.005$ for the following functions using MATLAB code.

(a) $y(t) = t^2 \exp(-2t) + \frac{t}{(t+1)^2}$

(b) $y(t) = \frac{\exp(-0.5t)}{(3t+2)^2}$

2.2 Calculate the numerical derivatives $\partial y(u_1, u_2, u_3)/\partial u_1$, $\partial y(u_1, u_2, u_3)/\partial u_2$, $\partial y(u_1, u_2, u_3)/\partial u_3$, $\partial^2 y(u_1, u_2, u_3)/\partial u_1^2$, $\partial^2 y(u_1, u_2, u_3)/\partial u_1 \partial u_2$ and $\partial^2 y(u_1, u_2, u_3)/\partial u_1 \partial u_3$ at $u_1 = 0.5$, $u_2 = 1.0$ and $u_3 = 0.0$ with $\Delta u_1 = 0.01$, $\Delta u_2 = 0.02$ and $\Delta u_3 = 0.005$ using MATLAB code.

(a) $y(u_1, u_2, u_3) = u_1^3 u_2 + u_2 u_3^2 + u_1 u_2 u_3$

(b) $y(u_1, u_2, u_3) = \exp(-u_1 u_2) + u_1^2 u_3 + \ln(u_1 u_2 + u_3)$

2.3 Simulate the following process using the Euler method with $\Delta t = 0.01$:

(a) $\frac{dy(t)}{dt} = -y(t) + (2 + 0.1y(t))u(t)$, $y(0) = 0.5$

$$u(t) = \begin{cases} 2 & t \geq 2.0 \\ 0 & t < 2.0 \end{cases}$$

(b) $\frac{dy(t)}{dt} = -y(t) + (2 + 0.1y(t))u(t - 0.3)$, $y(0) = 0.5$

$$u(t) = \begin{cases} 2 & t \geq 2.0 \\ 0 & t < 2.0 \end{cases}$$

(c) $\frac{d^4 y(t)}{dt^4} + 4 \frac{d^3 y(t)}{dt^3} + 6 \frac{d^2 y(t)}{dt^2} + 4 \frac{dy(t)}{dt} + y(t) = u(t)$, $u(t) = 1.1(1 - y(t))$,

$$\left. \frac{d^3 y(t)}{dt^3} \right|_{t=0} = \left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0$$

- (d) $\frac{d^4 y(t)}{dt^4} + 4 \frac{d^3 y(t)}{dt^3} + 6 \frac{d^2 y(t)}{dt^2} + 4 \frac{dy(t)}{dt} + y(t) = u(t), u(t) = 1.1(1 - y(t))$
 $+ \frac{1.1}{3.0} \int_0^t (1 - y(\tau)) d\tau, \quad \left. \frac{d^3 y(t)}{dt^3} \right|_{t=0} = \left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0$
- (e) $\frac{d^2 y(t)}{dt^2} + 3.0 \frac{dy(t)}{dt} + y(t) = 2u(t - 0.5), \quad u(t) = 0.7(1 - y(t)) + \frac{0.7}{2.0} \int_0^t (1 - y(\tau)) d\tau + 1,$
 $\left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t < 0$
- (f) $\frac{d^2 y(t)}{dt^2} + 3.0 \frac{dy(t)}{dt} + y(t) = 0.3 \frac{du(t - 0.5)}{dt} + 2u(t - 0.5), u(t) = 0.7(1 - y(t))$
 $+ \frac{0.7}{2.0} \int_0^t (1 - y(\tau)) d\tau + 1, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t < 0$
- (g) $\frac{d^2 y(t)}{dt^2} + 3.0 \frac{dy(t)}{dt} + y(t) = 0.3 \frac{du(t - 0.5)}{dt} + 2u(t - 0.5),$
 $u(t) = 0.7(y_s(t) - y(t)) + \frac{0.7}{2.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau + 0.3 \frac{d(y_s(t) - y(t))}{dt},$
 $y_s(t) = \begin{cases} 2 & t \geq 2.0 \\ 0 & t < 2.0 \end{cases}, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t \leq 0$
- (h) $\frac{d^2 y(t)}{dt^2} + 3.0 \frac{dy(t)}{dt} + y(t) = 0.3 \frac{du(t - 0.5)}{dt} + 2u(t - 0.5),$
 $u(t) = 0.7(y_s(t) - y(t)) + \frac{0.7}{2.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau + 0.3 \frac{d(y_s(t) - y(t))}{dt},$
 $y_s(t) = \begin{cases} 2 & t \geq 2.0 \\ 0 & t < 2.0 \end{cases}, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = -0.5, \quad y(0) = 0.2, \quad u(t) = 0.5 \quad \text{for } t < 0$
- (i) $\frac{d^2 y(t)}{dt^2} + 2.0 \frac{dy(t)}{dt} + y(t) = u(t - 0.5),$
 $u(t) = 1.5(y_s(t) - y(t)) + \frac{1.5}{3.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau + 1.5 \times 0.5 \frac{d(y_s(t) - y(t))}{dt} + d(t),$
 $\left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t < 0$
 $y_s(t) = \begin{cases} 1 & t \geq 1.0 \\ 0 & t < 1.0 \end{cases}, \quad d(t) = \begin{cases} -1 & t \geq 20.0 \\ 0 & t < 20.0 \end{cases}$

2.4 Estimate the following model for the given data using the least-squares method:

$$\text{Model } y = P_1x + P_2x^2$$

Data

x	1	2	3	4	5
y	5	16	33	56	85

2.5 Estimate the following model for the given data using the least-squares method:

$$\text{Model } y = P_1 \exp(-x) + P_2x^2$$

Data

x	1	2	3	4	5
y	3.74	12.27	27.10	48.04	75.01

2.6 Transform the following model to an appropriate form and estimate the model parameters using the least-squares method:

$$\text{Model } y = \frac{P_2x^3}{P_1 + x^3}$$

Data

x	1	2	3	4	5
y	1.000	1.778	1.929	1.969	1.984

2.7 Obtain the root of $y(x) = x^3 - 5x^2 + 5x - 8$ using the following methods:

- (a) bisection method with the initial interval between $x = 3$ and $x = 5$;
- (b) Newton–Raphson method using numerical derivative with the initial value $x_0 = 5.0$;
- (c) Newton–Raphson method using the analytic derivatives with the initial value $x_0 = 5.0$.

2.8 Obtain the roots of the following equations using the Newton–Raphson method with the initial values $x_1 = 1.0$ and $x_2 = 1.0$:

$$f_1(x_1, x_2) = x_1^2x_2^2 + \exp(-x_1)x_2 - 6.9829$$

$$f_2(x_1, x_2) = x_1 + x_1x_2^2 + \exp(-x_1 + 2.0)(x_2 + 3) - 17.842$$

- (a) use the numerical derivatives;
- (b) use the analytic derivatives.

2.9 Obtain the roots of the following equations using the Newton–Raphson method with the initial values $x_1 = 1.0$, $x_2 = 2.0$ and $x_3 = 3.0$:

$$f_1(x_1, x_2, x_3) = x_1^2x_2^2 + \exp(-x_1)x_2 + x_1x_3 - 9.3829$$

$$f_2(x_1, x_2, x_3) = x_1 + x_1x_2^2 + \exp(-x_1 + 2.0)(x_2 + 3) + (x_3 - 1)^2 - 18.842$$

$$f_3(x_1, x_2, x_3) = x_1 + x_2 + x_3 - 5.3000$$

2.10 Solve the following optimization problem using the interval halving method of which the initial interval is from 3 to 5:

$$\min_x (x^3 - 5x^2 + 5x - 8)^2$$

2.11 Solve the following optimization problem using the Levenberg–Marquardt method with the initial values $x_1 = 0.9$ and $x_2 = 1.5$:

$$\min_{x_1, x_2} \{F(x_1, x_2) = (x_1 - 1.0)^3 + (x_2 - 2.0)^2\}$$

2.12 Solve the following optimization problem using the Levenberg–Marquardt method with the initial values $P_1 = 0.9$ and $P_2 = 1.8$. Use the numerical derivatives to obtain the Jacobian.

Objective function :

$$\min_{P_1, P_2} \left\{ F(P_1, P_2) = \sum_{i=1}^5 \left(y_i - \frac{P_2 x_i^3}{P_1 + x_i^3} \right)^2 \right\}$$

Data

$x_1 = 1$	$x_2 = 2$	$x_3 = 3$	$x_4 = 4$	$x_5 = 5$
$y_1 = 1.000$	$y_2 = 1.778$	$y_3 = 1.929$	$y_4 = 1.969$	$y_5 = 1.984$

Bibliography

Kreyszig, E. (2006) *Advanced Engineering Mathematics*, John Wiley & Sons, Inc.

Ljung, L. (1987) *System Identification*, Prentice-Hall, Englewood Cliffs, New Jersey.

Reklaitis, G.V., Ravindran, A. and Ragsdell, K.M. (1983) *Engineering Optimization*, John Wiley & Sons, Inc.

Van Overschee, P. and De Moor, B. (1994) N4SID: subspace algorithms for the identification of combined deterministic–stochastic systems. *Automatica*, **30**, 75.

3

Dynamic Behavior of Linear Processes

3.1 Low-Order Plus Time-Delay Processes

The first-order plus time-delay (FOPTD) model and the second-order plus time-delay (SOPTD) model have been widely used to design and implement process controllers. And the terms related to the low-order plus time-delay processes are very useful for describing the dynamic characteristics of the process or the closed-loop control system. In this section, the dynamic behaviors of the FOPTD and SOPTD processes for the step input are derived and the important terms are introduced.

3.1.1 First-Order Plus Time-Delay Processes

The FOPTD process is

$$G(s) = \frac{y(s)}{u(s)} = \frac{k \exp(-\theta s)}{\tau s + 1} \quad (3.1)$$

where k , τ and θ are called the static gain, the time constant and the time delay respectively. The step input response (3.3) can be obtained by solving the differential equation (3.1) for the initial conditions (3.2):

$$u(t) = d \quad \text{for } t \geq 0 \quad \text{and} \quad u(t) = 0 \quad \text{for } t < 0, \quad y(t) = 0 \quad \text{for } t \leq 0 \quad (3.2)$$

$$y(t) = \left[1 - \exp\left(-\frac{t-\theta}{\tau}\right) \right] kd \quad \text{for } t \geq \theta, \quad y(t) = 0 \quad \text{for } t < \theta \quad (3.3)$$

The following equations can be obtained from (3.3):

$$y(\infty) = \left[1 - \exp\left(-\frac{t-\theta}{\tau}\right) \right] kd \Big|_{t=\infty} = kd \quad (3.4)$$

$$y(t)|_{t=\theta+\tau} = \left[1 - \exp\left(-\frac{t-\theta}{\tau}\right) \right] kd \Big|_{t=\theta+\tau} = [1 - \exp(-1)]kd = 0.6321y(\infty) \quad (3.5)$$

$$\frac{dy(t)}{dt} \Big|_{t=\theta} = \exp\left(-\frac{t-\theta}{\tau}\right) \frac{kd}{\tau} \Big|_{t=\theta} = \frac{kd}{\tau} = \frac{y(\infty)}{\tau}, \quad \frac{dy(t)}{dt} = 0 \quad \text{for } t < \theta \quad (3.6)$$

Note that the first derivative of the process output is affected directly by the process input. So, the first derivative $dy(t)/dt$ has a discontinuity at $t = \theta$. Figure 3.1 brings this all together.

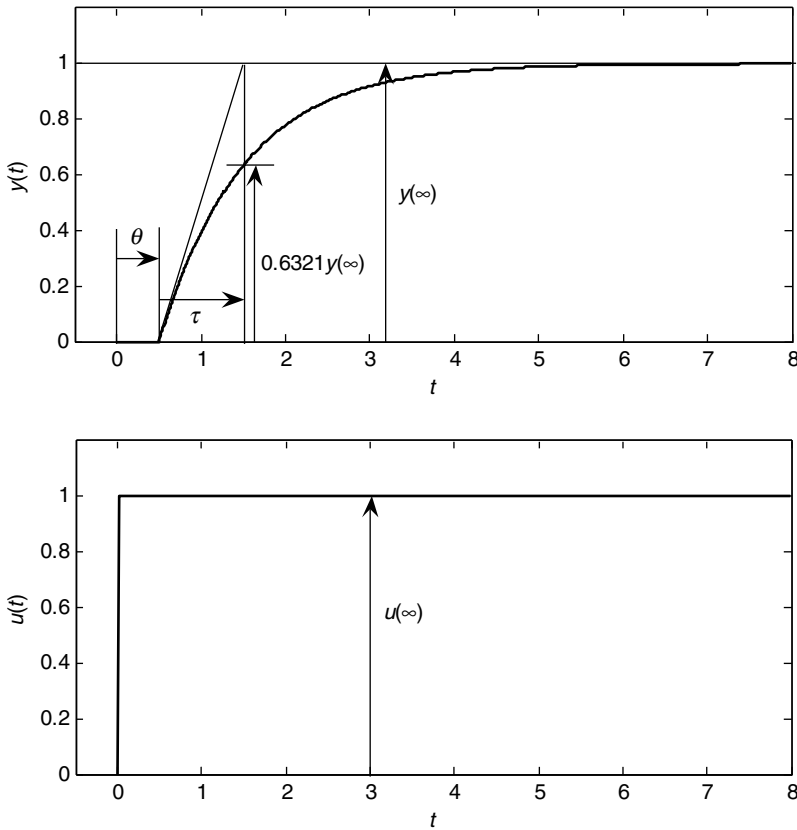


Figure 3.1 Step response of a FOPTD process.

As shown in Figure 3.1, the physical meaning of the time constant is how fast the process responds to the process input. The static gain represents how much the process output changes for the variation of the process input. The time delay is the time required for the process input to affect the process output for the first time.

Example 3.1

The process shows the step response of Figure 3.2 for the step input from 0 to 2 at $t = 0$. Obtain the FOPTD model.

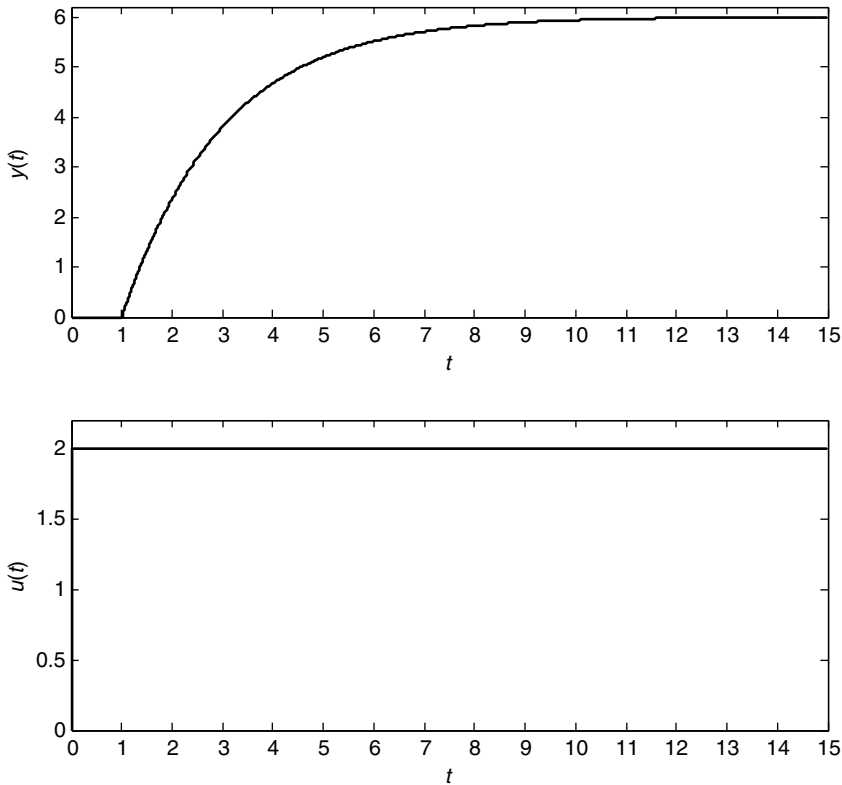


Figure 3.2 Response of a FOPTD process for the step input test of Example 3.1.

Solution The static gain is $k = y(\infty)/d = 6.0/2.0 = 3.0$. The time delay is $\theta = 1.0$ directly from Figure 3.2. The time constant $\tau = 2.0$ is obtained by drawing the tangent line at $t = 1.0$ and reading the point at which the tangent line and the $y(t) = 6.0$ line intersect. The same result is obtained by reading the time at which the process output reaches $0.6321y(\infty)$. So, the FOPTD model obtained is $G(s) = 3.0 \exp(-1.0s)/(2.0s + 1)$.

Note that this kind of process identification is only applicable to the FOPTD process. More general approaches will be introduced later.

3.1.2 Second-Order Plus Time-Delay Processes

The SOPTD process is

$$G(s) = \frac{y(s)}{u(s)} = \frac{k \exp(-\theta s)}{\tau^2 s^2 + 2\tau\xi s + 1} \quad (3.7)$$

where k , τ , ξ and θ are called the static gain, the time constant, the damping factor and the time delay respectively. The step input response can be obtained by solving the differential equation (3.7) for the initial conditions (3.8):

$$u(t) = d \quad \text{for } t \geq 0 \quad \text{and} \quad u(t) = 0 \quad \text{for } t < 0, \quad y(t) = 0 \quad \text{for } t < 0 \quad (3.8)$$

For $\xi > 1$ (overdamped process)

$$y(t) = kd \left\{ 1 - \frac{\tau_1 \exp[-(t-\theta)/\tau_1] - \tau_2 \exp[-(t-\theta)/\tau_2]}{\tau_1 - \tau_2} \right\} \quad \text{for } t \geq \theta \quad (3.9)$$

$$y(t) = 0 \quad \text{for } t < \theta \quad (3.10)$$

For $\xi = 1$ (critically damped process)

$$y(t) = kd \{ 1 - [1 + (t-\theta)/\tau] \exp[-(t-\theta)/\tau] \} \quad (3.11)$$

$$y(t) = 0 \quad \text{for } t < \theta \quad (3.12)$$

For $\xi < 1$ (underdamped process)

$$y(t) = kd \left(1 - \exp[-\xi(t-\theta)/\tau] \left\{ \cos \left[\frac{\sqrt{1-\xi^2}}{\tau} (t-\theta) \right] + \frac{\xi}{\sqrt{1-\xi^2}} \sin \left[\frac{\sqrt{1-\xi^2}}{\tau} (t-\theta) \right] \right\} \right) \quad (3.13)$$

$$y(t) = 0 \quad \text{for } t < \theta \quad (3.14)$$

where

$$\tau_1 = \frac{\tau}{\xi - \sqrt{\xi^2 - 1}}, \quad \tau_2 = \frac{\tau}{\xi + \sqrt{\xi^2 - 1}}$$

Note that $d^2y(t)/dt^2$ is affected directly by the process input. But $dy(t)/dt$ is affected indirectly, because it is a result of the integral of $d^2y(t)/dt^2$. So, the first derivative $dy(t)/dt$ has no discontinuity at $t = \theta$. The second derivative has a discontinuity at $t = \theta$.

As shown in Figure 3.3, there are the three types of step response for the SOPTD process. One is the overdamped process, which shows no oscillation and the roots of the denominator $\tau^2 s^2 + 2\tau\xi s + 1 = 0$ are distinct and real (equivalently, $\xi > 1$). Another is the critically damped process, which shows no oscillation and the denominator $\tau^2 s^2 + 2\tau\xi s + 1 = 0$ has

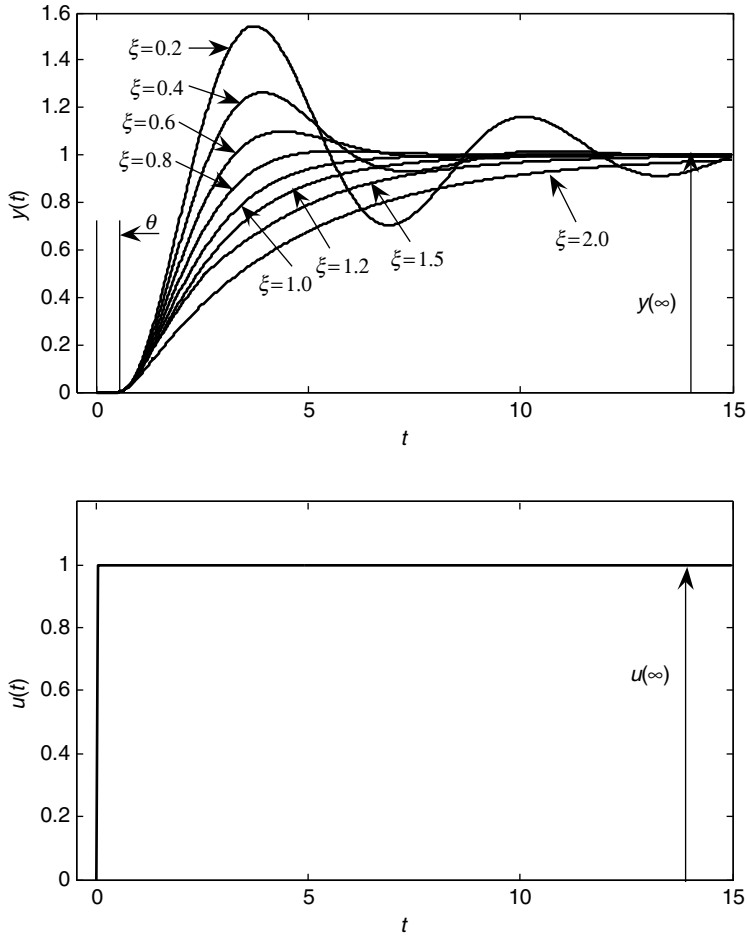


Figure 3.3 Responses of a SOPTD process with respect to the damping factor.

the real double root (equivalently, $\xi = 1$). The other is the underdamped process, which shows oscillation and the roots of the denominator $\tau^2 s^2 + 2\tau\xi s + 1 = 0$ are distinct and complex (equivalently, $\xi < 1$).

In particular, consider the underdamped response in Figure 3.4. The following terms to characterize the underdamped response should be kept in mind, because they have been widely used to characterize the control performance of the closed-loop control system.

t_{p1} and t_{p2} are the first peak time and the second peak time. The settling time t_s is the time required for the process output to reach within 5% of the final value. The overshoot and the decay ratio are defined as $(y_{p1} - y(\infty))/y(\infty)$ and $(y_{p2} - y(\infty))/(y_{p1} - y(\infty))$ respectively.

Obtain the following equations from (3.13):

$$t_p = \theta + \pi\tau / \sqrt{1 - \xi^2} = \theta + P/2 \quad \text{time to first peak} \quad (3.15)$$

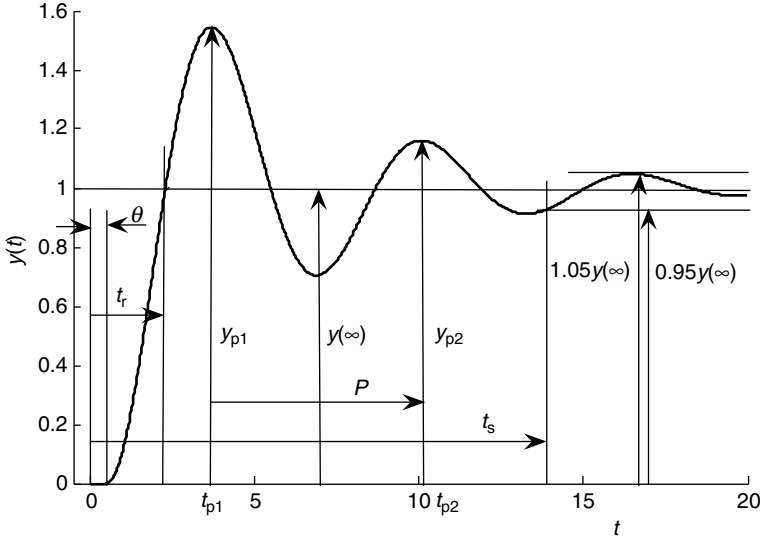


Figure 3.4 Underdamped response.

$$P = \frac{2\pi\tau}{\sqrt{1-\xi^2}} \quad \text{period} \quad (3.16)$$

$$\frac{y_{p1} - y(\infty)}{y(\infty)} = \exp\left(\frac{-\pi\xi}{\sqrt{1-\xi^2}}\right) \quad \text{overshoot} \quad (3.17)$$

$$\frac{y_{p2} - y(\infty)}{y_{p1} - y(\infty)} = \left(\frac{y_{p1} - y(\infty)}{y(\infty)}\right)^2 = \exp\left(\frac{-2\pi\xi}{\sqrt{1-\xi^2}}\right) \quad \text{decay ratio} \quad (3.18)$$

Example 3.2

Figure 3.5 shows the step response (top) for a process receiving a step input (bottom) from 0 to 1 at $t = 0$. Obtain the SOPTD model.

Solution The static gain is $k = y(\infty)/d = 1.0/1.0 = 1.0$. The time delay is $\theta = 1.0$ directly from Figure 3.5. And the first peak is $y_{p1} = 1.37$ and the period $P = 9.8$ from Figure 3.5. Then, $\xi = 0.3$ and $\tau = 1.5$ are obtained from $(y_{p1} - y(\infty))/y(\infty) = \exp[-\pi\xi/(1-\xi^2)^{0.5}]$ and $P = 2\pi\tau/(1-\xi^2)^{0.5}$. So, the SOPTD model obtained is $G(s) = 1.0 \exp(-1.0s)/(1.5^2s^2 + 2.0 \times 1.5 \times 0.3s + 1)$.

3.2 Process Reaction Curve Method

The process reaction curve (PRC) method is used to obtain the FOPTD model from the step input test. The procedure is as follows. First, a step input u_∞ enters until the process output reaches a steady state y_∞ , as shown in Figure 3.6. Note that most processes in industry show step

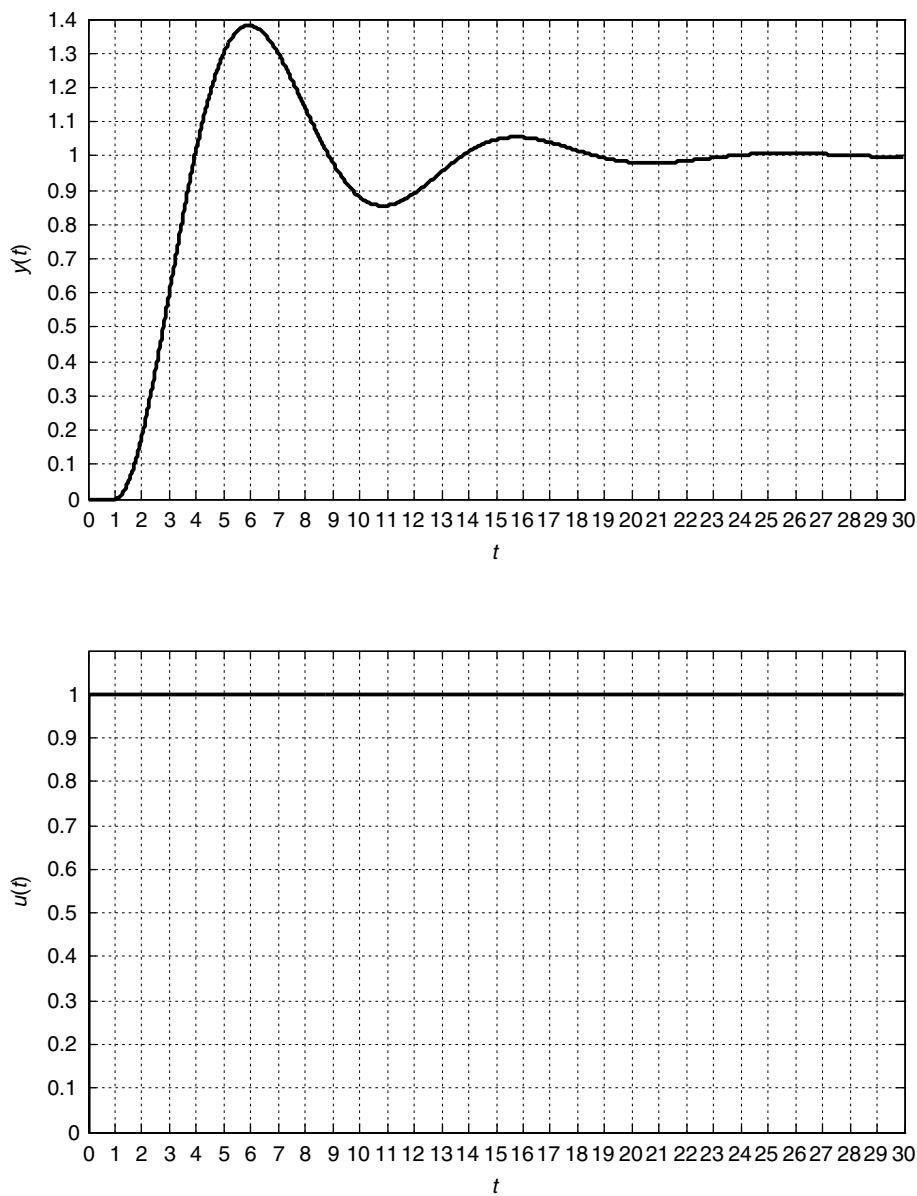


Figure 3.5 Step response of a SOPTD process for the step test of Example 3.2.

responses like Figure 3.6 rather than those of Figure 3.2 because they are not the FOPTD (process orders are higher than unity in most cases).

Second, draw the tangent line at the inflection point. Third, determine the time delay θ and the time constant τ , as shown in Figure 3.6. Fourth, the static gain can be estimated by the ratio of the process output to the step input; that is, $k = y_{\infty}/u_{\infty}$. The PRC method can be easily understood by comparing Figures 3.6 and 3.1. The method is very simple and provides the exact

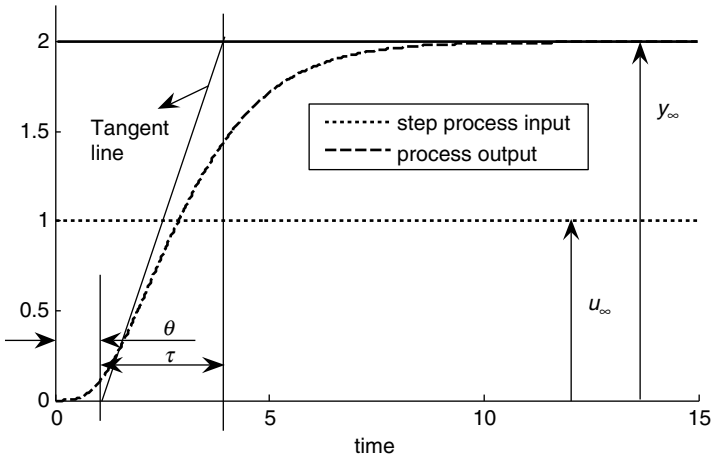


Figure 3.6 Typical step response of the usual open-loop stable and overdamped processes.

model for the FOPTD process. However, it also suffers from several practical problems. The FOPTD model has limitations in approximating underdamped/high-order processes. Also, it is difficult to determine the inflection point if the measurements are contaminated by measurement noise. Moreover, the method requires a long identification time and the model obtained can be sensitive to disturbances and/or noises because it opens the control loop until the system reaches the steady state.

Example 3.3

Figure 3.7 shows the step response (top) for a process receiving a step input (bottom) from 0 to 1 at $t = 0$. Obtain the FOPTD model using the PRC method.

Solution The static gain is $k = y(\infty)/d = 1.0/1.0 = 1.0$. The time delay $\theta = 1.0$ and the time constant $\tau = 4.0$ are obtained by drawing the tangent line at the inflection point and reading the point at which the tangent line and the $y(t) = 1.0$ line intersect. So, the FOPTD model obtained is $G(s) = 1.0 \exp(-1.0s)/(4.0s + 1)$.

3.3 Poles and Zeroes

Poles are finite s values that make the transfer function infinite. Zeroes are finite s values that make the transfer function zero. For example, the transfer function (1.162) has a pole of -1 and a zero of $-1/3$. Roughly speaking, poles make the denominator of the transfer function zero and zeroes make the numerator of the transfer function zero. The dynamic characteristics can be understood by investigating the poles and the zeroes as follows.

3.3.1 Relationship Between Poles and Dynamic Behaviors

Consider the following examples to see how the dynamic characteristics are connected with poles. The results of the examples will be summarized to derive extremely important conclusions on the stability.

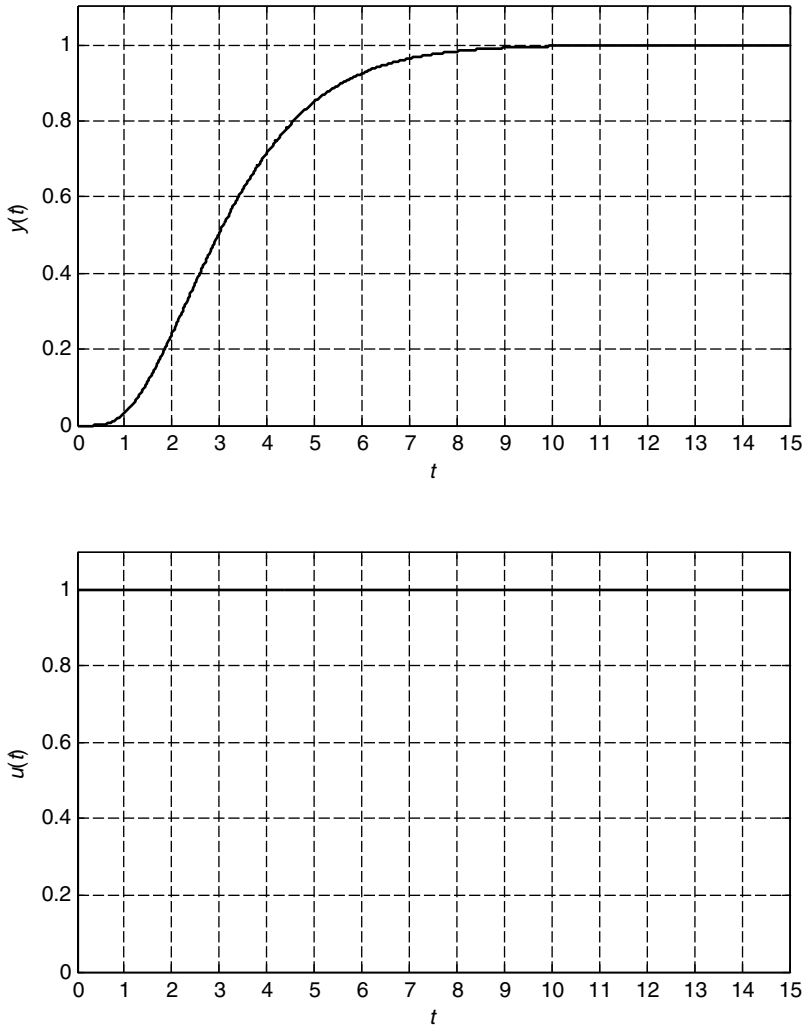


Figure 3.7 Step response of a process for the step test.

Example 3.4

The process has real positive poles.

$$\frac{y(s)}{u(s)} = \frac{1}{(s-1)(s-2)} \quad (3.19)$$

The process in (3.19) has poles of 1 and 2. Equation (3.19) can be rewritten for the step input signal $u(s) = 1/s$:

$$y(s) = \frac{1}{(s-1)(s-2)s} = \frac{1/2}{s} - \frac{1}{s-1} + \frac{1/2}{s-2}, \quad y(t) = \frac{1}{2} - \exp(t) + \frac{1}{2}\exp(2t) \quad (3.20)$$

As shown in Example 3.4, the process output $y(t)$ diverges exponentially as t increases because it is a linear combination of the monotonically increasing exponential functions $\exp(t)$ and $\exp(2t)$, originated from the positive poles.

Example 3.5

The process has real negative poles.

$$\frac{y(s)}{u(s)} = \frac{1}{(s+1)(s+2)} \quad (3.21)$$

The process (3.21) has poles of -1 and -2 . Equation (3.21) can be rewritten for the step input signal $u(s) = 1/s$:

$$y(s) = \frac{1}{(s+1)(s+2)} = \frac{1/2}{s} - \frac{1}{s+1} + \frac{1/2}{s+2}, \quad y(t) = \frac{1}{2} - \exp(-t) + \frac{1}{2}\exp(-2t) \quad (3.22)$$

As shown in Example 3.5, the process output $y(t)$ converges exponentially to the steady-state value of $1/2$ as t increases because it is a linear combination of the monotonically decreasing exponential functions $\exp(-t)$ and $\exp(-2t)$, originated from the negative poles.

Example 3.6

The process has complex poles of which the real parts are negative.

$$\frac{y(s)}{u(s)} = \frac{2}{s^2 + 2s + 2} = \frac{2}{[s - (-1 + i)][s - (-1 - i)]} \quad (3.23)$$

The process (3.23) has complex poles of $-1 \pm i$. Equation (3.23) can be rewritten for the step input signal $u(s) = 1/s$:

$$y(s) = \frac{2}{s[s - (-1 + i)][s - (-1 - i)]} = \frac{1}{s} - \frac{(1-i)/2}{s - (-1 + i)} - \frac{(1+i)/2}{s - (-1 - i)} \quad (3.24)$$

$$\begin{aligned} y(t) &= 1 - \frac{1-i}{2}\exp(-t+it) - \frac{1+i}{2}\exp(-t-it) \\ &= 1 - \frac{\exp(-t)}{2}[(1-i)\exp(it) + (1+i)\exp(-it)] \end{aligned} \quad (3.25)$$

Equation (3.25) can be rewritten using the Euler formula $\exp(ix) = \cos(x) + i \sin(x)$:

$$y(t) = 1 - \exp(-t)(\cos t + \sin t) \quad (3.26)$$

As shown in Example 3.6, the magnitude of the process output $y(t)$ converges exponentially to the steady-state value of 1 as t increases because the magnitude is determined by the monotonically decreasing exponential function $\exp(-t)$, originated from the real part of the pole. Also, $y(t)$ oscillates with a period of 2π because of the two functions $\exp(it)$ and $\exp(-it)$, originated from the imaginary part of the pole.

Example 3.7

The process has complex poles of which the real parts are positive.

$$\frac{y(s)}{u(s)} = \frac{2}{s^2 - 2s + 2} = \frac{2}{[s - (1 + i)][s - (1 - i)]} \quad (3.27)$$

The process (3.27) has complex poles of $1 \pm i$. Equation (3.27) can be rewritten for the step input signal $u(s) = 1/s$:

$$\frac{y(s)}{u(s)} = \frac{2}{s[s - (1 + i)][s - (1 - i)]} = \frac{1}{s} - \frac{(1 - i)/2}{s - (1 + i)} - \frac{(1 + i)/2}{s - (1 - i)} \quad (3.28)$$

$$y(t) = 1 - \frac{1 - i}{2} \exp(t + it) - \frac{1 + i}{2} \exp(t - it) = 1 - \frac{\exp(t)}{2} [(1 - i) \exp(it) + (1 + i) \exp(-it)] \quad (3.29)$$

Equation (3.29) can be rewritten using the Euler formula $\exp(ix) = \cos(x) + i \sin(x)$:

$$y(t) = 1 - \exp(t)(\cos t + \sin t) \quad (3.30)$$

As shown in Example 3.7, the magnitude of the process output $y(t)$ diverges exponentially as t increases because the magnitude is determined by the monotonically increasing exponential function $\exp(t)$, originated from the real part of the pole. Also, $y(t)$ oscillates with a period of 2π because of the two functions $\exp(it)$ and $\exp(-it)$, originated from the imaginary part of the pole.

Example 3.8

Consider the following strictly proper system in which the poles are different from each other. Strictly proper means that the order of the numerator is less than that of the denominator.

$$G(s) = \frac{a(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)}, \quad n > m \quad (3.31)$$

where p_k and z_k are the poles and zeroes of the system and a is an arbitrary constant. The poles and the zeroes are real or complex numbers. Equation (3.31) for a step input can be rewritten equivalently by partial fractions as follows:

$$y(s) = \frac{A_0}{s} + \frac{A_1}{s - p_1} + \frac{A_2}{s - p_2} + \cdots + \frac{A_n}{s - p_n} \quad (3.32)$$

Then, $y(t) = A_0 + A_1 \exp(p_1 t) + A_2 \exp(p_2 t) + \cdots + A_n \exp(p_n t)$ is obtained. That is, $y(t)$ is a linear combination of $\exp(p_k t)$, $k = 1, 2, \dots, n$, originated from the poles. By the Euler formula:

$$\exp(p_k t) = \exp(\operatorname{Re}(p_k)t + i \operatorname{Im}(p_k)t) = \exp(\operatorname{Re}(p_k)t) [\cos(\operatorname{Im}(p_k)t) + i \sin(\operatorname{Im}(p_k)t)]$$

$\operatorname{Re}(p_k)$ and $\operatorname{Im}(p_k)$ are the real part and the imaginary part respectively. Note that $\exp(\operatorname{Re}(p_k)t)$ determines the magnitude. So, if one pole has a positive real part, then the system is

unstable. If all the poles have negative real parts then all the terms are exponentially converging functions, so that the system is stable for a step input.

The same conclusion can be reached for the case that some of the poles are the same, like double root, triple root and so on. If the process is rewritten with partial fractions for the case that it has a double root of p_k , then it will contain $t \exp(p_k t)$ additionally. So, the stability is still determined by the real part of the pole.

Also, the system is unstable for a step input if one of the poles is located on zero because the partial fraction form of $y(s)$ contains $1/s^2$, which is an unstable ramp function.

In summary, the following statements can be concluded by generalizing the results of Examples 3.4–3.8.

1. If all the poles are real, then the process output will not oscillate for a step input signal.
2. If one of the poles is complex, then the process output will oscillate for a step input signal.
3. If one of the poles has a positive real part the process output will diverge (unstable) for a step input signal.
4. If the poles have negative real parts the process output will not diverge (stable) for a step input signal.
5. If one of the poles is located on zero, the process output will diverge for a step input signal.

3.3.2 Stable Poles, Unstable Poles, Left-Half-Plane Pole, Right-Half-Plane Pole

The pole is called a right-half-plane (RHP) pole (or unstable pole) if the real part of the pole is positive. Similarly, if the real part of the pole is negative, then it is called a left-half-plane (LHP) pole (or stable pole). When locating on the pole in Figure 3.8, it can be understood why they are termed thus.

Here, $\text{Re}(p)$ and $\text{Im}(p)$ denote the real part and the imaginary part of the pole. The LHP means the left part of the y -axis in Figure 3.8. The $-1.5 + 1.0i$ and $-1.5 - 1.0i$ poles belong to the LHP. The $1.5 + 1.0i$ and $1.5 - 1.0i$ poles belong to the RHP. Because the RHP pole has a positive real part, it is called the unstable pole. The LHP pole has a negative real part. Thus, it is called the stable pole.

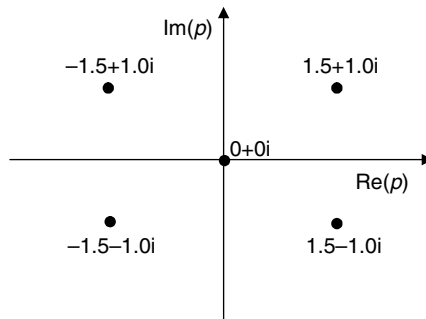


Figure 3.8 Pole positions.

3.3.3 Open-Loop Stable and Unstable Processes

If the process output is stable and converges to a constant value for a step process input, then it is called an open-loop stable process. Otherwise, it is called an open-loop unstable process. If the process has a transfer function in which the poles are located on zero, then it is called an integrating process. And the process is called an unstable process if it has a transfer function in which some of the poles have positive real parts. For example, $G(s) = \exp(-0.1s)/[s(s + 1)]$ and $G(s) = \exp(-0.1s)/[s^3(s + 1)(s + 0.1)]$ are integrating processes and $G(s) = \exp(-0.1s)/[(s + 1)(10s - 1)]$ and $G(s) = \exp(-0.2s)/[(s - 0.1 + 0.01i)(s - 0.1 - 0.01i)]$ are unstable processes. The integrating process and the unstable process are open-loop unstable processes.

3.3.4 Relationship Between Zeroes and Dynamic Behaviors

Zeroes determine the initial dynamics of the process output for an abrupt change of process input. Attention needs paying to the fact that the numerator of the transfer function corresponds to the differentials of the process input. If there is an abrupt change in the process input, then the differential terms in the numerator become large. Meanwhile, the differential terms are negligible for a smooth process input. Consider the following examples.

Example 3.9

Case 1 has the large zero of 10, Case 2 has a small positive zero of 0.1 and Case 3 has a small negative zero of -0.1 .

$$\text{Case 1 : } \frac{y(s)}{u(s)} = \frac{-0.1s + 1}{s^2 + 2s + 1} \quad (3.33)$$

$$\text{Case 2 : } \frac{y(s)}{u(s)} = \frac{-10s + 1}{s^2 + 2s + 1} \quad (3.34)$$

$$\text{Case 3 : } \frac{y(s)}{u(s)} = \frac{10s + 1}{s^2 + 2s + 1} \quad (3.35)$$

The numerators of the three cases correspond to the differentials $-0.1du(t)/dt + u(t)$, $-10du(t)/dt + u(t)$ and $10du(t)/dt + u(t)$ respectively. Assume that the process input $u(t)$ changes from zero to a positive value abruptly. Then, $du(t)/dt$ will be a large positive value at the instant of the abrupt change and $du(t)/dt$ will go back to zero after the instant. Consider the simulation in Figure 3.9. The process input $u(t)$ is a step signal ($u(t) = 1$ for $t \geq 0$, $u(t) = 0$ for $t < 0$).

Note that $-0.1du(t)/dt + u(t)$, $-10du(t)/dt + u(t)$ and $10du(t)/dt + u(t)$ of the three cases are the same for $t > 0$ because $du(t)/dt = 0$ at $t > 0$. So, although the initial responses are totally different from each other, the late parts of the three step-responses converge on each other. The positive value of $10du(t)/dt$ at $t = 0$ in Case 3 is much bigger than $-0.1du(t)/dt$ at $t = 0$ in Case 1, so that the process output of Case 3 deviates positively from the process output of Case 1. The big negative value of $-10du(t)/dt$ at $t = 0$ in Case 2 drags the process output down initially. In Case 2, the initial direction of the process output is opposite to the direction of the late part. This is called the inverse response.

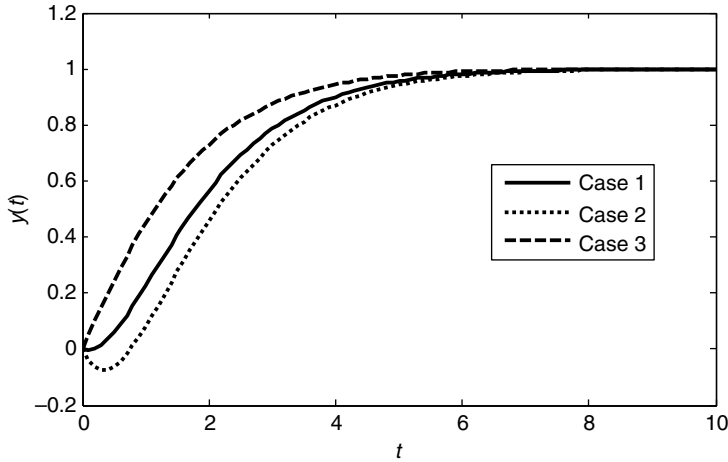


Figure 3.9 Step responses of Case 1, Case 2 and Case 3.

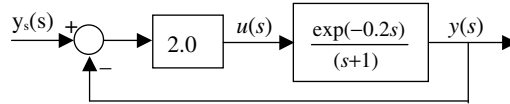
In summary, a small positive zero produces an inverse response or drag down of the process output initially for a positive step input. A small negative zero initially drags up the process output for a positive step input.

3.4 Block Diagram

A block diagram is a useful tool to show signal flows in a systematic way between the transfer functions (blocks). Consider the following examples to understand the relationships between the block diagram and the transfer functions and several important properties of the block diagram.

Example 3.10

The following block diagram is equivalent to Equations (3.36)–(3.38):



The input and the output of the first transfer function (block) are $y_s(s) - y(s)$ and $u(s)$ respectively. The transfer function of the first block is 2.0; that is, $u(s)/y_s(s) - y(s) = 2.0$. So:

$$u(s) = 2.0(y_s(s) - y(s)) \Leftrightarrow u(t) = 2.0(y_s(t) - y(t)) \quad (3.36)$$

In a similar way, $u(s)$ and $y(s)$ are the input and the output of the second transfer function (that is, $y(s)/u(s) = \exp(-0.2s)/(s + 1)$ respectively. Then, (3.37) is derived:

$$sy(s) + y(s) = u(s)\exp(-0.2s) \Leftrightarrow \frac{dy(t)}{dt} + y(t) = u(t - 0.2) \quad (3.37)$$

From (3.36) and (3.37):

$$sy(s) + y(s) = 2.0(y_s(s) - y(s))\exp(-0.2s) \Leftrightarrow \frac{dy(t)}{dt} + y(t) = 2.0(y_s(t - 0.2) - y(t - 0.2)) \quad (3.38)$$

Meanwhile, the transfer function from $y_s(s)$ to $y(s)$ can be derived in a straightforward way. By multiplying $y(s)/u(s) = \exp(-0.2s)/(s + 1)$ and $u(s)/(y_s(s) - y(s)) = 2$ the transfer function from $y_s(s) - y(s)$ to $y(s)$ is obtained as follows:

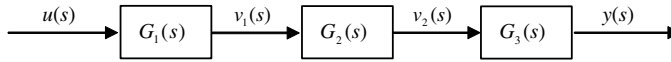
$$\frac{y(s)}{y_s(s) - y(s)} = \frac{2 \exp(-0.2s)}{s + 1} \quad (3.39)$$

Equation (3.39) can be rewritten to the following transfer function from $y_s(s)$ to $y(s)$.

$$\frac{y(s)}{y_s(s)} = \frac{2 \exp(-0.2s)/(s + 1)}{1 + 2 \exp(-0.2s)/(s + 1)} \quad (3.40)$$

Example 3.11

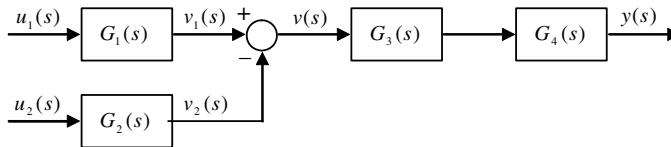
Obtain the overall transfer function from $u(s)$ to $y(s)$ for the following block diagram, in which several transfer functions are connected sequentially.



Solution From the block diagram, $v_1(s)/u(s) = G_1(s)$, $v_2(s)/v_1(s) = G_2(s)$ and $y(s)/v_2(s) = G_3(s)$. Then, it is straightforward to obtain $y(s)/u(s) = G_1(s)G_2(s)G_3(s)$ by multiplying the three terms.

Example 3.12

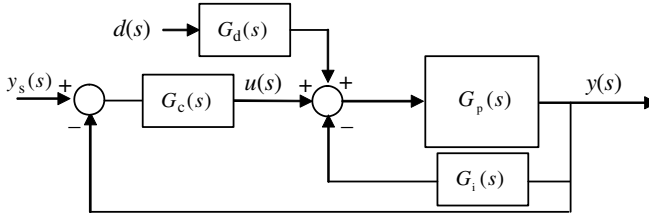
Obtain the overall transfer function from $u_1(s)$ and $u_2(s)$ to $y(s)$ for the following block diagram, in which the outputs of the two blocks are added, followed by the two transfer functions:



Solution From the block diagram, $v_1(s)/u_1(s) = G_1(s)$, $v_2(s)/u_2(s) = G_2(s)$, $v_1(s) - v_2(s) = v(s)$ and $y(s)/v(s) = G_3(s)G_4(s)$. That is, $v_1(s) = G_1(s)u_1(s)$, $v_2(s) = G_2(s)u_2(s)$ and $y(s) = G_3(s)G_4(s)(v_1(s) - v_2(s))$. Then, it is straightforward to obtain $(G_1(s)u_1(s) - G_2(s)u_2(s))G_3(s)G_4(s) = y(s)$.

Example 3.13

Obtain the transfer functions from $d(s)$ to $y(s)$, from $y_s(s)$ to $y(s)$ and from $d(s)$ and $y_s(s)$ to $y(s)$ for the following block diagram:



Solution Let us assume $y_s(s) = 0$ to obtain the transfer function from $d(s)$ to $y(s)$. Then, $(-y(s)G_c(s) + d(s)G_d(s) - y(s)G_i(s))G_p(s) = y(s)$ can be obtained in a straightforward manner by considering Examples 3.11 and 3.12. This can be rewritten as follows:

$$\frac{y(s)}{d(s)} = \frac{G_p(s)G_d(s)}{1 + G_p(s)G_i(s) + G_p(s)G_c(s)} \quad (3.41)$$

Let us assume $d(s) = 0$ to obtain the transfer function from $y_s(s)$ to $y(s)$. Then, $((y_s(s) - y(s))G_c(s) - y(s)G_i(s))G_p(s) = y(s)$ is obtained in a straightforward manner. That is:

$$\frac{y(s)}{y_s(s)} = \frac{G_p(s)G_c(s)}{1 + G_p(s)G_i(s) + G_p(s)G_c(s)} \quad (3.42)$$

Using the superposition rule, the transfer function from $d(s)$ and $y_s(s)$ to $y(s)$ is obtained by adding the two transfer functions (3.41) and (3.42):

$$y(s) = \frac{G_p(s)G_d(s)}{1 + G_p(s)G_i(s) + G_p(s)G_c(s)}d(s) + \frac{G_p(s)G_c(s)}{1 + G_p(s)G_i(s) + G_p(s)G_c(s)}y_s(s) \quad (3.43)$$

3.5 Frequency Responses

The frequency responses have been widely used in the research area of process control and process identification. Estimating the frequency responses of the process from the process data is one of the most important things in process identification. Also, analyzing techniques on the basis of the frequency responses have played an important role in designing process controllers and analyzing the stability of the closed-loop control system. In this section, the frequency response is defined and the relationship between the frequency response and the transfer function is discussed. The ultimate frequency and the ultimate gain are then defined, which are extremely important for proportional–integral–derivative (PID) controller tuning. Finally, the Bode plot and the Nyquist plot for graphic representations of the frequency responses are introduced.

3.5.1 Frequency Responses of Linear Processes

Consider the following important fact. When you enter a sine signal input $u(t) = a \sin(\omega t)$ into a linear process for a long time, the process output always becomes a sine signal of the same frequency, like $y(t) = b \sin(\omega t + c)$, as shown in Figure 3.10. In this case, the ratio of the amplitude of the process output to that of the process input (i.e. b/a) is called the amplitude ratio

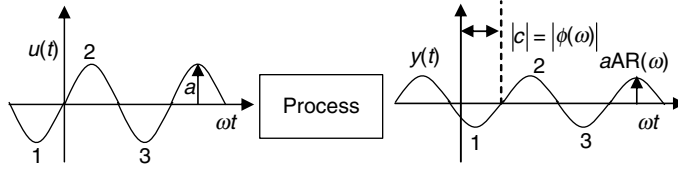


Figure 3.10 Typical response of a process output for a sine signal.

of the process (denoted by $AR(\omega)$) and the phase difference c between the process output and the process input is called the phase angle of the process (denoted by $\phi(\omega)$). In other words, the process output is $y(t) = aAR(\omega) \sin(\omega t + \phi(\omega))$ for the process input $u(t) = a \sin(\omega t)$ in cyclic-steady-state. The set of the amplitude ratio $AR(\omega)$ and the phase angle $\phi(\omega)$ is called the frequency response of the process. The frequency response is a function of the frequency ω .

Usually, the process output is lagged backward so that $\phi(\omega)$ is negative. Also, the magnitude of the phase angle $|\phi(\omega)|$ becomes bigger and the amplitude ratio becomes smaller monotonically as the frequency of the sine input signal increases.

Example 3.14

Assume that you use a sine signal to activate the process. After entering the process input $u(t) = 5.2 \sin(0.5t)$ for a long time, you find the process output $y(t) = 1.3 \sin(0.5t - \pi/6)$. In this case, the amplitude ratio $AR(\omega)$ is 0.25 and the phase angle $\phi(\omega)$ is $-\pi/6$ at a frequency of 0.5.

Example 3.15

Assume that you perform two experiments using two different sine signals. In the first experiment, you find the process output $y(t) = 1.5 \sin(0.5t - \pi/10)$ after entering the process input $u(t) = 3.0 \sin(0.5t)$ for a long time. Next, you perform the other experiment with $u(t) = 6.0 \sin(1.0t)$ and find $y(t) = 1.5 \sin(1.0t - \pi/4)$. In this case, two frequency responses of the process are obtained: $AR(0.5) = 0.5$, $\phi(0.5) = -\pi/10$ and $AR(1.0) = 0.25$, $\phi(1.0) = -\pi/4$.

3.5.2 Estimating Frequency Responses from the Transfer Function

The frequency response of the process can be obtained directly from the transfer function without simulation or plant test. Assume that $G(s)$ of the transfer function of the process is available. Then, the amplitude ratio and the phase angle can be estimated by the following equations.

$$AR(\omega) = |G(i\omega)| = \sqrt{\text{Re}(G(i\omega))^2 + \text{Im}(G(i\omega))^2} \quad (3.44)$$

$$\phi(\omega) = \angle G(i\omega) = \arctan \frac{\text{Im}(G(i\omega))}{\text{Re}(G(i\omega))} \quad \text{for } \text{Re}(G(i\omega)) \text{ and } \text{Im}(G(i\omega)) \quad (3.45)$$

Equations (3.44) and (3.45) are equivalent to the complex number (3.46) because $\exp(i \angle G(i\omega)) = \cos(\angle G(i\omega)) + i \sin(\angle G(i\omega))$.

$$G(i\omega) = |G(i\omega)| \exp(i \angle G(i\omega)) \quad (3.46)$$

Now, $G(i\omega)$ can be calculated from (3.46) if the amplitude ratio and the phase angle are given.

Proof

Assume $G(s)$ is stable and a strictly proper transfer function and the structure of the transfer function is

$$G(s) = \frac{k(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)}, \quad n > m \quad (3.47)$$

where all poles are distinct. For the process input $u(t) = a \sin(\omega t)$, of which the Laplace transform is $u(s) = a\omega/(s^2 + \omega^2)$, the process output will be

$$y(s) = \frac{a\omega G(s)}{(s - i\omega)(s + i\omega)} = \frac{ka\omega(s - z_1)(s - z_2) \cdots (s - z_m)}{(s - p_1)(s - p_2) \cdots (s - p_n)(s - i\omega)(s + i\omega)} \quad (3.48)$$

Then, (3.49) is obtained by the partial fraction

$$y(s) = \frac{A_1}{s - p_1} + \frac{A_2}{s - p_2} + \cdots + \frac{A_n}{s - p_n} + \frac{aG(i\omega)}{2i} \frac{1}{s - i\omega} + \frac{aG(-i\omega)}{-2i} \frac{1}{s + i\omega} \quad (3.49)$$

where A_k is a constant. So, the process output in the time domain is

$$y(t) = A_1 \exp(p_1 t) + \cdots + A_n \exp(p_n t) + \frac{aG(i\omega)}{2i} \exp(i\omega t) - \frac{aG(-i\omega)}{2i} \exp(-i\omega t) \quad (3.50)$$

Note that $G(s)$ is stable, so the real parts of all the poles except $i\omega$ and $-i\omega$ are negative. Then, all the terms except $\exp(-i\omega t)$ and $\exp(i\omega t)$ will decay to zero after a long time. As a result, the final process output is

$$y(t) = \frac{aG(i\omega)}{2i} \exp(i\omega t) - \frac{aG(-i\omega)}{2i} \exp(-i\omega t) \quad (3.51)$$

Note that $G(i\omega) = \text{Re}(G(i\omega)) + i \text{Im}(G(i\omega))$ means $G(-i\omega) = \text{Re}(G(i\omega)) - i \text{Im}(G(i\omega))$, resulting in $|G(i\omega)| = |G(-i\omega)|$ and $\angle G(i\omega) = -\angle G(-i\omega)$. From the three equations $G(i\omega) = |G(i\omega)| \exp(i \angle G(i\omega))$, $\angle G(i\omega) = -\angle G(-i\omega)$ and $|G(i\omega)| = |G(-i\omega)|$, (3.52) is obtained.

$$y(t) = \frac{a|G(i\omega)|}{2i} \exp(i\omega t + i \angle G(i\omega)) - \frac{a|G(i\omega)|}{2i} \exp(-i\omega t - i \angle G(i\omega)) \quad (3.52)$$

This can be rewritten as (3.53) by using the Euler formula $\sin(x) = (\exp(ix) - \exp(-ix))/(2i)$:

$$y(t) = a|G(i\omega)| \sin(\omega t + \angle G(i\omega)) \quad (3.53)$$

Equation (3.53) proves (3.44) and (3.45). That is, if the process is strictly proper and stable then the process output becomes $y(t) = a|G(i\omega)| \sin(\omega t + \angle G(i\omega))$ for the process input $u(t) = a \sin(\omega t)$. This means that the amplitude ratio is $|G(i\omega)|$ and the phase angle is $\angle G(i\omega)$. The proof can easily be extended to the case that some poles are multiple roots.

Note that the frequency response for the zero frequency ($\omega = 0$) is $G(0)$ from $G(i\omega)$. And, note that $G(0)$ is the static gain (that is, $G(0) = y_{ss}/u_{ss}$) for the open-loop stable process because setting $s = 0$ means steady state (that is, all the derivatives are zero). For example, $G(0)$ is $k = y_{ss}/u_{ss}$ for the FOPTD process of $G(s) = k \exp(-\theta s)/(\tau s + 1)$. So, the frequency response for the zero frequency is the same as the static gain for the open-loop stable process.

Example 3.16

Assume that $G(s) = G_1(s)G_2(s) \cdots G_n(s)$ and $G_k(i\omega)$, $k = 1, 2, \dots, n$, are given. Then, $|G(i\omega)|$ and $\angle G(i\omega)$ can be easily obtained from the following equations:

$$|G(i\omega)| = |G_1(i\omega)| |G_2(i\omega)| \cdots |G_n(i\omega)| \quad (3.54)$$

$$\angle G(i\omega) = \angle G_1(i\omega) + \angle G_2(i\omega) + \cdots + \angle G_n(i\omega) \quad (3.55)$$

This can be derived directly by using the representation $G_k(i\omega) = |G_k(i\omega)| \exp(i\angle G_k(i\omega))$.

Example 3.17

Predict the process output in the case that you enter the process input $u(t) = 2.0 \sin(1.0t)$ into the linear time-invariant process of which the transfer function is $G(s) = \exp(-0.1s)/(s + 2)^2$.

Solution Because $|G(i1.0)| = 0.2$ and $\angle G(i1.0) = -1.027$, $y(t) = 0.4 \sin(1.0t - 1.027)$.

Example 3.18

Predict the process output in the case that you enter the process input $u(t) = 2.0 \sin(1.0t) + 3 \sin(1.5t)$ into the process of which the transfer function is $G(s) = \exp(-0.1s)/(s + 2)^2$.

Solution Because $|G(i1.0)| = 0.2$ and $\angle G(i1.0) = -1.027$, $y(t) = 0.4 \sin(1.0t - 1.027)$ for $u(t) = 2.0 \sin(1.0t)$. And $y(t) = 0.48 \sin(1.5t - 1.437)$ for $u(t) = 3.0 \sin(1.5t)$ because $|G(i1.5)| = 0.16$ and $\angle G(i1.5) = -1.437$. By the superposition rule, the solution is $y(t) = 0.4 \sin(1.0t - 1.027) + 0.48 \sin(1.5t - 1.437)$ for $u(t) = 2.0 \sin(1.0t) + 3 \sin(1.5t)$.

Example 3.19

You obtained $y(t) = 0.3 \sin(0.5t - \pi/8)$ for $u(t) = 1.0 \sin(0.5t)$ from an experiment. Find $G(0.5i)$ for the process.

Solution The amplitude ratio $AR(0.5) = G(0.5i) = 0.3$ and the phase angle $\phi(0.5) = \angle G(0.5i) = -\pi/8$ are obtained from the experiment. So, the solution is $G(0.5i) = |G(0.5i)| \exp(i\angle G(0.5i)) = 0.3 \exp(-i\pi/8) = 0.277 - i0.115$.

Example 3.20

You obtained $y(t) = 0.8 \sin(0.5t - \pi/8) + 0.5 \sin(1.0t - \pi/4) + 0.1 \sin(3.0t - 3\pi/4) + 1.0$ for $u(t) = \sin(0.5t) + \sin(1.0t) + \sin(3.0t) + 1.0$ from an experiment. Find $G(0.5i)$, $G(1.0i)$, $G(3.0i)$ and $G(0.0)$ for the process.

Solution By the superposition rule, the process would show $y(t) = 0.8 \sin(0.5t - \pi/8)$ for $u(t) = \sin(0.5t)$, $y(t) = 0.5 \sin(1.0t - \pi/4)$ for $u(t) = \sin(1.0t)$, $y(t) = 0.1 \sin(3.0t - 3\pi/4)$ for $u(t) = \sin(3.0t)$ and $y(t) = 1.0$ for $u(t) = 1.0$. So, $G(0.5i) = 0.8 \exp(-i\pi/8)$, $G(1.0i) = 0.5 \exp(-i\pi/4)$, $G(3.0i) = 0.1 \exp(-i3\pi/4)$ and $G(0.0) = 1.0/1.0$.

3.5.3 Ultimate Gain (Ratio) and Ultimate Frequency

Assume that you enter a sine signal $a \sin(\omega t)$ into a process for a long time and then the process output becomes a sine signal of the same frequency ω . In this case, if the process output is lagged by $-\pi$ (that is, $y(t) = b \sin(\omega t - \pi) = -b \sin(\omega t)$ for $u(t) = \sin(\omega t)$, and equivalently $\angle G(i\omega) = -\pi$) for the frequency ω , as shown in Figure 3.11, then the frequency is the ultimate frequency ω_u . Also, the ultimate gain is $k_{cu} = a/b$, which is the reciprocal of the amplitude ratio ($k_{cu} = 1/AR(\omega_u)$ or $k_{cu} = 1/|G(i\omega_u)|$) for the ultimate frequency.

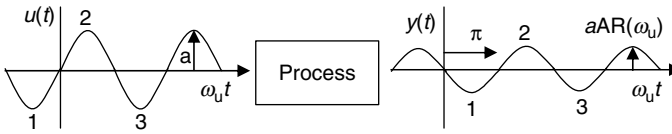


Figure 3.11 Response of a process output for a sine signal of ultimate frequency ω_u .

Example 3.21

After performing many experiments changing the frequency, you find that the process output is $y(t) = -1.3 \sin(2.2t)$ for the process input $u(t) = 2.6 \sin(2.2t)$. Then, the ultimate frequency, the ultimate period and the ultimate gain of the process are 2.2, $2\pi/2.2$ and 2.0 respectively.

Example 3.22

The ultimate frequency can be estimated directly by solving $\angle G(i\omega_u) = -\pi$ (and equivalently $\text{Im}(G(i\omega_u)) = 0$) if the transfer function $G(s)$ of the process is given. Then, it is straightforward to estimate the ultimate gain $k_{cu} = 1/|G(i\omega_u)|$. Find the ultimate frequency, the ultimate period, the ultimate amplitude ratio and the ultimate gain for the process of which the transfer function is $G(s) = \exp(-0.2s)/(s + 2)^2$.

Solution The MATLAB code to estimate the ultimate frequency data using the bisection method and the results are shown in Table 3.1. The bisection method estimates the ultimate frequency by finding the root of $\text{Im}(G(i\omega_u)) = 0$.

3.5.4 Bode Plot and Nyquist Plot

A Bode plot is a set of two graphs of the amplitude ratio and the phase angle with respect to frequency. A Nyquist plot is a graph in which the x -axis is the real part and the y -axis the imaginary part of the frequency response $G(i\omega)$.

Example 3.23

Let us draw the Bode plot and Nyquist plot for the following processes:

$$G_1(s) = \frac{1}{2s+1}, \quad G_2(s) = \frac{1}{(2s+1)^2}, \quad G_3(s) = \frac{1}{(2s+1)^3} \quad (3.56)$$

$$G_4(s) = \exp(-0.2s), \quad G_5(s) = \frac{\exp(-0.2s)}{2s+1} \quad (3.57)$$

From $G_1(i\omega) = 1/(2i\omega + 1) = (-2i\omega + 1)/(4\omega^2 + 1) = 1/(4\omega^2 + 1) + i(-2\omega)/(4\omega^2 + 1)$ we have $\text{Re}(G_1(i\omega)) = 1/(4\omega^2 + 1)$ and $\text{Im}(G_1(i\omega)) = (-2\omega)/(4\omega^2 + 1)$. Then, it is straightforward to draw the Nyquist plot by plotting $\text{Re}(G_1(i\omega))$ versus $\text{Im}(G_1(i\omega))$ with changing frequency ω , as shown in Figure 3.12. Here, PA denotes the phase angle.

Table 3.1 MATLAB code to estimate the ultimate frequency data using the bisection method in Example 3.22.

uf_ex2.m	g_uf_ex2.m
<pre>clear; w=0.0; delta_w=0.01; g_L=imag(g_uf_ex2(delta_w)); %imaginary part while (1) % obtain the boundary for the bisection method w=w+delta_w; g_R=imag(g_uf_ex2(w)); if (g_R*g_L < 0) break; end end w_L = delta_w; w_R = w; % boundary while (1) % bisection method w_M = (w_L+w_R)/2.0; g_M=imag(g_uf_ex2(w_M)); g_R=imag(g_uf_ex2(w_R)); if (g_M*g_R > 0) w_R=w_M; else w_L=w_M; end if (abs(w_R-w_L)<0.000000001) break; end end wu=w_M; ARu=abs(real(g_uf_ex2(w_M))); Pu=2*pi/wu; Ku=1/ARu; fprintf('wu = %7.5f Pu = %7.5f \n',wu,Pu); fprintf('ARu = %7.5f Ku = %7.5f \n',ARu,Ku);</pre>	<pre>function [g]=g_uf_ex2(w) s=i*w; g=exp(-0.2*s)/(s+2)^2; end</pre> <p>command window</p> <pre>>> uf_ex2 wu = 4.32841 Pu = 1.45162 ARu = 0.04398 Ku = 22.73511</pre>

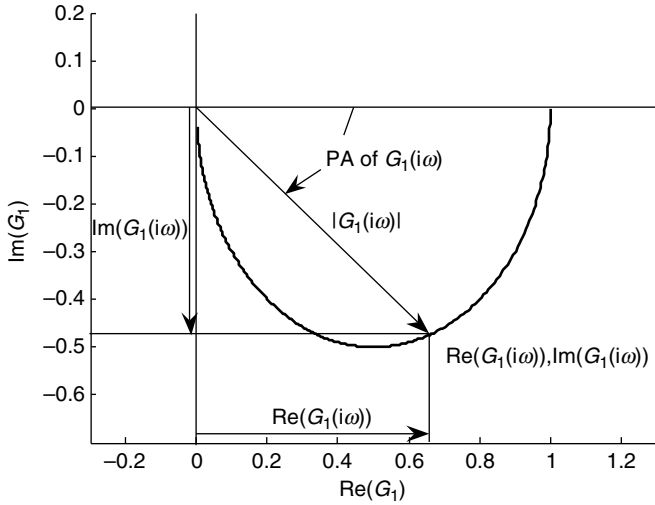


Figure 3.12 The Nyquist plot of $G_1(i\omega)$.

$\angle G_1(i\omega) = \tan^{-1}(\text{Im}/\text{Re}) = -\tan^{-1}(2\omega)$ and $|G_1(i\omega)| = 1/|2i\omega + 1| = 1/\sqrt{4\omega^2 + 1}$. Now, it is straightforward to draw the Bode plot, as shown in Figure 3.13. Here, the scales of the x -axis and the y -axis of the amplitude ratio plot in the Bode plot are $\log_{10}|G_1(i\omega)|$ and $\log_{10}\omega$. The scales of the x -axis and the y -axis of the phase-angle plot in the Bode plot are $\angle G_1(i\omega)$ and $\log_{10}\omega$.

Note that $|G_3(i\omega)| = |1/(2i\omega + 1)| |1/(2i\omega + 1)| |1/(2i\omega + 1)| = 1/(\sqrt{4\omega^2 + 1})^3$ and $\angle G_3(i\omega) = 3\angle G_1(i\omega) = -3\tan^{-1}(2\omega)$. Thus, the Nyquist plot and the Bode plot are as shown in Figures 3.14 and 3.15.

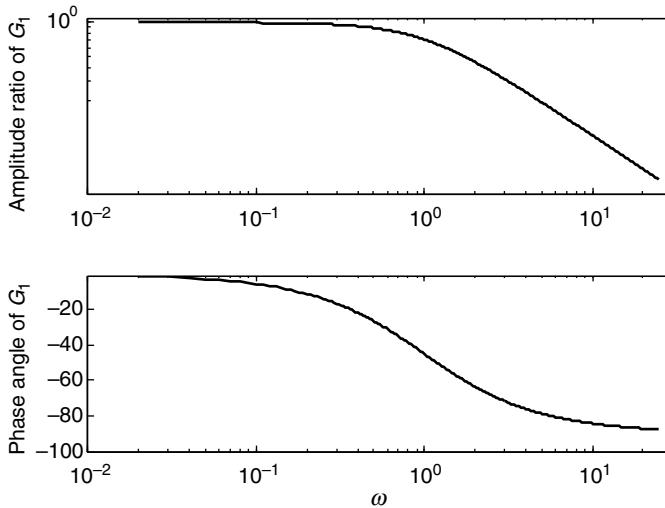


Figure 3.13 The Bode plot of $G_1(\omega)$.

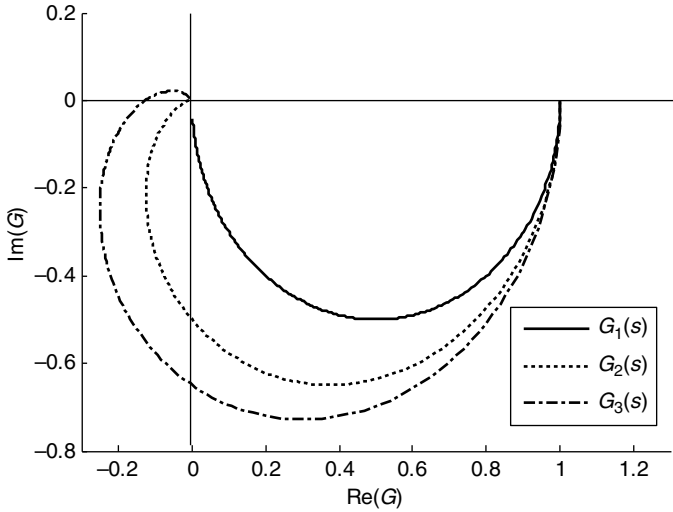


Figure 3.14 The Nyquist plot of $G_1(i\omega)$, $G_2(i\omega)$ and $G_3(i\omega)$.

$|G_4(i\omega)| = |\exp(-i0.2\omega)| = 1$ and $\angle G_4(i\omega) = -0.2\omega$ are obtained for the fourth process. Also, $|G_5(i\omega)| = |\exp(-i0.2\omega)|/|i2\omega + 1| = 1/\sqrt{4\omega^2 + 1}$ and $\angle G_5(i\omega) = -0.2\omega - \tan^{-1}(2\omega)$ for the fifth process. Thus, the Nyquist plot and the Bode plot are as shown in Figures 3.16 and 3.17.

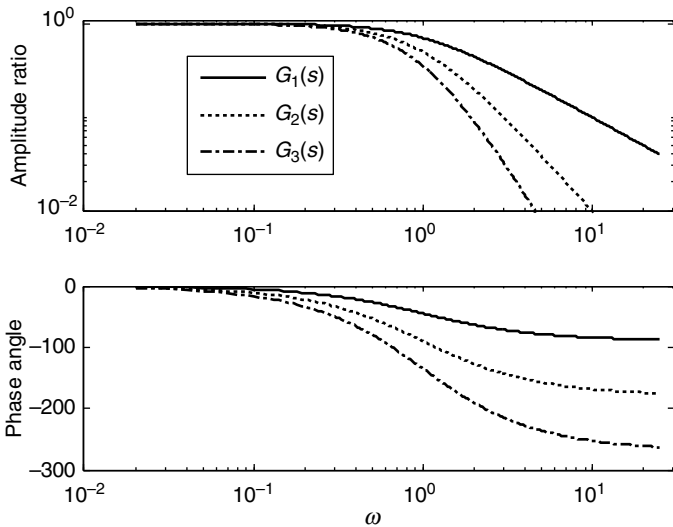


Figure 3.15 The Bode plot of $G_1(i\omega)$, $G_2(i\omega)$ and $G_3(i\omega)$.

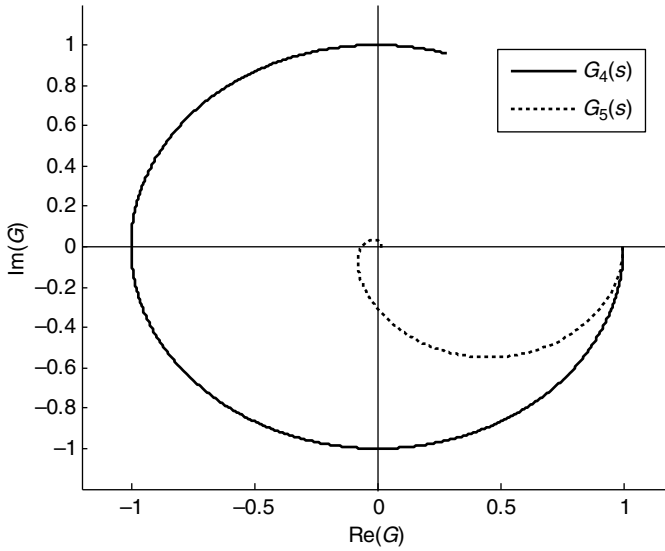


Figure 3.16 The Nyquist plot of $G_4(i\omega)$ and $G_5(i\omega)$.

Example 3.24

The ultimate frequency of $G_3(s)$ can be guessed from the Bode plot of Figure 3.15 by checking the frequency corresponding to the phase angle of -180° . Then, it is straightforward to measure the ultimate gain of $G_3(s)$ by checking the amplitude ratio corresponding to the ultimate frequency. The ultimate gain can also be measured from the Nyquist plot in a similar way. $G_1(s)$ and $G_2(s)$ have no ultimate frequency because they never cross the phase angle of -180° .

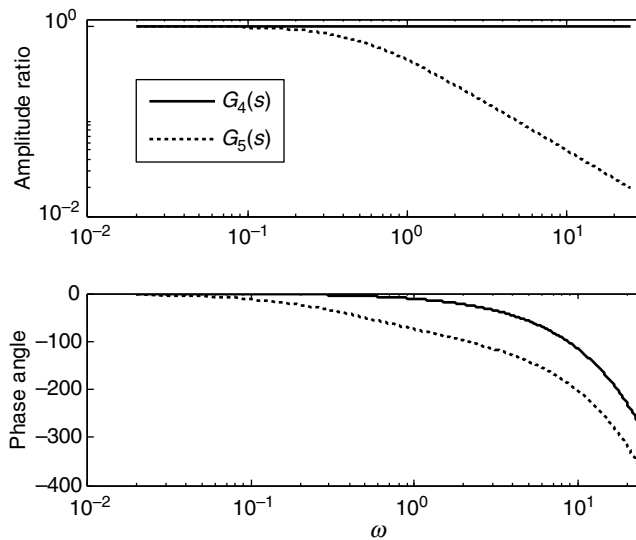


Figure 3.17 The Bode plot of $G_4(i\omega)$ and $G_5(i\omega)$.

Example 3.25

Using a computer program, the plots can be drawn in a very simple way. Program the MATLAB code to draw the Bode and Nyquist plots for $G_5(s) = \exp(-0.2s)/(2s + 1)$.

Solution The MATLAB code to draw the Bode plot and the Nyquist plot of $G_5(i\omega)$ is shown in Table 3.2. Here, $\arctan 2(\text{Im}, \text{Re})$ returns the phase angle of the complex number of $\text{Re} + i \text{Im}$. It returns the phase angle ranged from $-\pi$ and $+\pi$ while $\arctan(\text{Im}/\text{Re})$ returns the phase angle ranged from $-\pi/2$ and $+\pi/2$.

Problems

- 3.1 Explain the following terms:
- (a) time constant, time delay, static gain, FOPTD model;

(b) damping factor, underdamped process, overdamped process, overshoot, decay ratio, settling time, SOPTD model;

(c) step input response, PRC method;

(d) poles, zeroes, transfer function, inverse response;

(e) block diagram, amplitude ratio, phase angle, frequency response;

(f) ultimate gain, ultimate frequency, ultimate amplitude ratio;

(g) Bode plot, Nyquist plot.

Table 3.2 MATLAB code to draw the Bode plot and the Nyquist plot of $G_5(i\omega)$.

<div>bode_nyquist_ex3.m</div> <div>clear; w_max=10.0; delw=0.01; dummy_pab=0.0; pa_base=0.0; n=round(w_max/delw); s=0*i; m=1; G=g_bn_ex3(s); R(m)=real(G); I(m)=imag(G); W(m)=0; AR(m)=abs(G); for m=2:n s=i*m*delw; G=g_bn_ex3(s); W(m)=m*delw; R(m)=real(G); I(m)=imag(G); if(I(m) ≥ 0.0 & I(m-1)<0.0) pa_base=pa_base- 2*pi; end pa=pa_base+atan2(I(m),R(m)); PA(m)=pa*180/pi; AR(m)=abs(G); end figure(1); subplot(2,1,1); loglog(W,AR); %Bode subplot(2,1,2); semilogx(W,PA); figure(2); plot(R,I); %Nyquist</div>	<div>g_bn_ex3.m</div> <div>function [g]=g_bn_ex3(s) g=exp(-0.2*s)/(s+2)^2; end</div> <div>command window</div> <div>>> bode_nyquist_ex3</div>
--	---

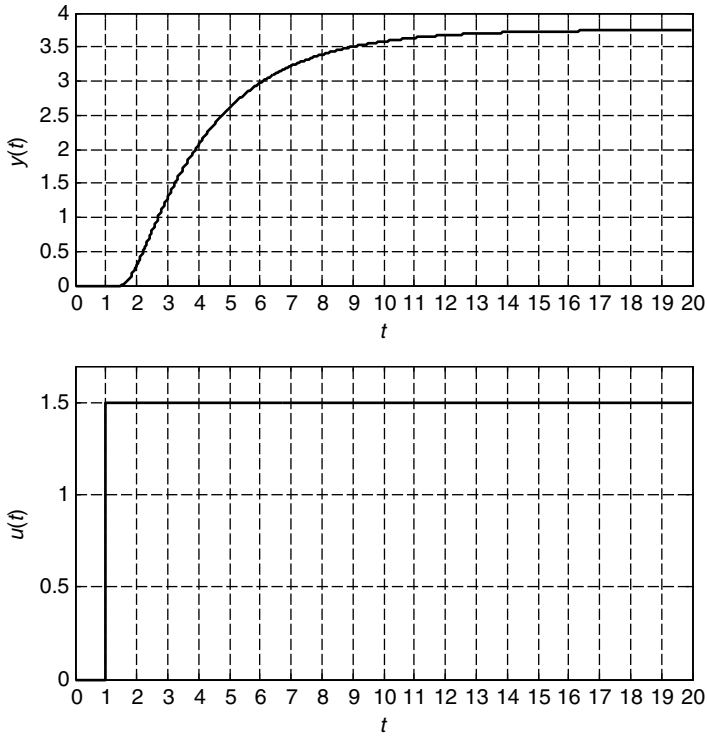


Figure P3.1

- 3.2 Estimate the FOPTD model for the step response shown in Figure P3.1.
- 3.3 Estimate the SOPTD model for the step response shown in Figure P3.2.
- 3.4 Run the real-time virtual processes (refer to Appendix for details) and choose Process 1 and code the step test using a MATLAB m-file and perform the step input test to Process 1. Now estimate the FOPTD model to fit the step response of Process 1.
- 3.5 Perform the step input test to the virtual process Process 2 (refer to Appendix for details) and estimate the SOPTD model to fit the dynamics of Process 2.
- 3.6 Perform the step input test to the virtual process of Process 3 (refer to Appendix for details) and estimate the FOPTD model for Process 3. Note that the process output and the process input are not zero initially. So, you need to define new deviation variables for the process input and output.
- 3.7 Solve Problem 3.6 again with Process 4 (refer to Appendix for details) and the SOPTD model.
- 3.8 Find the poles and the zeroes for the following processes:

$$(a) \quad \frac{y(s)}{u(s)} = \frac{s - 0.5}{(s + 1)(s + 2)}$$

$$(b) \quad \frac{y(s)}{u(s)} = \frac{s + 0.5}{(s^2 + s + 1)(s + 1)}$$

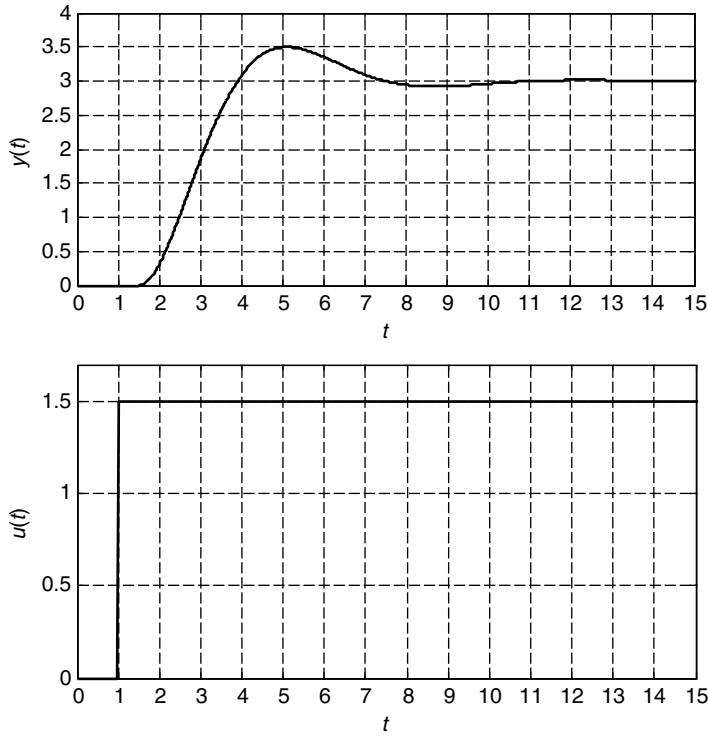


Figure P3.2

$$(c) \quad \frac{y(s)}{u(s)} = \frac{1}{s(s+1)^2(s+3)}$$

3.9 Find all the processes that show unstable response, stable response, oscillatory response, nonoscillatory response and inverse response respectively for a step process input among the following processes:

$$(a) \quad \frac{y(s)}{u(s)} = \frac{10(s+5)(s+10)}{(s+1)(s+2)(s+3)(s+4)}$$

$$(b) \quad \frac{y(s)}{u(s)} = \frac{\exp(-0.1s)}{s^2 + 0.5s + 1}$$

$$(c) \quad \frac{y(s)}{u(s)} = \frac{(0.5s+1)\exp(-0.1s)}{s^2 - s + 1}$$

$$(d) \quad \frac{y(s)}{u(s)} = \frac{-10s+2}{(s^2+2s+1)(2s+1)}$$

$$(e) \quad \frac{y(s)}{u(s)} = \frac{5s+1}{s(2s+1)(3s+1)}$$

$$(f) \quad \frac{y(s)}{u(s)} = \frac{-5s+1}{s^3(2s+1)(3s+1)}$$

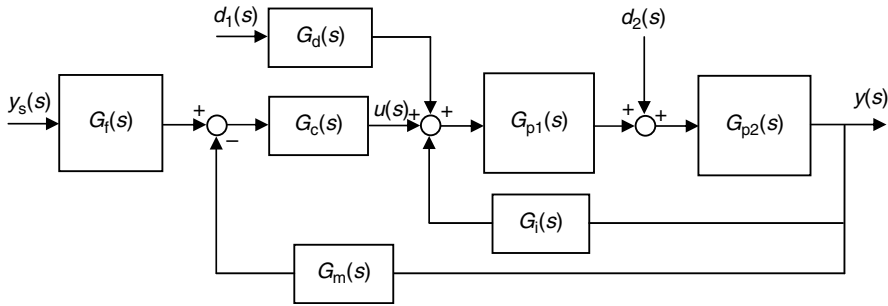


Figure P3.3

(g)
$$\frac{y(s)}{u(s)} = \frac{\exp(-0.3s)}{(s+1)(s+2)(s-1)}$$

3.10 Consider the block diagram in Figure P3.3.

- Find the transfer function from $y_s(s)$ to $y(s)$.
- Find the transfer function from $d_1(s)$ to $y(s)$.
- Find the transfer function from $d_2(s)$ to $y(s)$.
- Find the relationship between $y_s(s)$, $d_1(s)$, $d_2(s)$ and $y(s)$ using the above-obtained transfer functions.

3.11 You obtain the process output $y(t) = 0.5 \sin(2t - \pi/2)$ for the process input $u(t) = 2 \sin(2t)$ for a given process. Find the process outputs of the given process for the following process inputs:

- $u(t) = 4 \sin(2t)$
- $u(t) = 2 \sin(2t - \pi/4)$
- $u(t) = \sin(2t + \pi/2)$.

3.12 You perform two sine input experiments. In the first experiment, $y(t) = 0.25 \sin(2t - 3\pi/4)$ is obtained for $u(t) = \sin(2t)$. Then $y(t) = 0.5 \sin(t - \pi/4)$ for $u(t) = \sin(t)$ is obtained in the second experiment. Find $G(1i)$ and $G(2i)$. Here, $G(s)$ is the transfer function of the process.

3.13 You perform a step input test and a sine input test. $y(t) = 2$ is obtained for $u(t) = 1$ from the step test and $y(t) = 0.7 \sin(3t)$ for $u(t) = \sin(3t + \pi/6)$ is obtained from the sine test. Find $G(0i)$ and $G(3i)$.

3.14 You obtain the process output $y(t) = 2 + (2/\sqrt{2})\sin(2t - 3\pi/4)$ for the process input $u(t) = 1 + \sin(2t)$ for a given process. The process is an FOPTD process described by $\tau(dy(t)/dt) + y(t) = ku(t - \theta)$. Find the parameters τ , k and θ .

3.15 Obtain the process output $y(t)$ for the process input $u(t) = 0.5 \sin(t + \pi/4) + 2 \sin(2t - \pi/4) + 3$. The transfer function of the process satisfies $G(0i) = 5$, $G(1i) = -1 - i$ and $G(2i) = -0.5$.

3.16 The ultimate frequency and the ultimate gain of a process are 1.5 and 5.0 respectively. Find $G(1.5i)$. Here, $G(s)$ is the transfer function of the process.

- 3.17 Find the ultimate frequency, the ultimate period and the ultimate gain for the process $G(s) = 1/(s + 1)^5$.
- 3.18 Draw the Bode plots and the Nyquist plots for the following processes and select all the processes that have the ultimate frequency:

(a) $G(s) = \frac{1}{s + 1}$

(b) $G(s) = \frac{1}{(10s + 1)^2}$

(c) $G(s) = \frac{1}{(s + 1)^3}$

(d) $G(s) = \frac{10\exp(-0.1s)}{(10s + 1)^2}$

(e) $G(s) = \exp(-s)$.

Bibliography

- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
- Stephanopoulos, G. (1984) *Chemical Process Control – An Introduction to Theory and Practice*, Prentice-Hall.

Part Two

Process Control

Process control is about making the process output behave in the desired way by manipulating the process input in an automatic way. Process controllers have contributed much to improving the quality of products and reducing utility consumption. In this part, the PID controller and the tuning methods are introduced in Chapters 4 and 5. The two chapters are extremely important to understanding the basic concepts of feedback process control and various real-world controllers implemented in industry. Chapter 6 introduces several important tools (Bode plot and Nyquist plot) and terms (characteristic equation, critical frequency and gain, gain margin and phase margin) to analyze/describe the closed-loop dynamic characteristics of the designed feedback controller. Enhanced control strategies using additional measurements and the process model, which have been widely used in industry, are discussed in Chapter 7.

4

Proportional–Integral–Derivative Control

PID controllers have been most widely used in industry due to their simplicity, good control performance and excellent robustness to uncertainties. In this chapter, the structure of the PID controller and the roles of the three parts (proportional, integral and derivative) of the PID controller are discussed, followed by the practical issues related to the PID controller, such as the integral windup, implementation and industrial versions of the controller.

4.1 Structure of Proportional–Integral–Derivative Controllers and Implementation in Computers/Microprocessors

In this section, the structure of the PID controller is explained and their implementation in computers or microprocessors is discussed.

4.1.1 Structure of Proportional–Integral–Derivative Controllers

PID controllers are composed of the following three parts:

$$\text{Proportional (P) part : } u_P(t) = k_c(y_s(t) - y(t)) \quad (4.1)$$

$$\text{Integral (I) part : } u_I(t) = \frac{k_c}{\tau_i} \int_0^t (y_s(\tau) - y(\tau)) d\tau \quad (4.2)$$

$$\text{Derivative (D) part : } u_D(t) = k_c \tau_d \frac{d(y_s(t) - y(t))}{dt} \quad (4.3)$$

The output of the PID controller is the sum of the above-mentioned three parts:

$$u(t) = u_P(t) + u_I(t) + u_D(t) = k_c(y_s(t) - y(t)) + \frac{k_c}{\tau_i} \int_0^t (y_s(\tau) - y(\tau)) d\tau + k_c \tau_d \frac{d(y_s(t) - y(t))}{dt} \quad (4.4)$$

where $y_s(t)$, $y(t)$ and $u(t)$ denote the setpoint (the desired process output), the process output and the control output of the PID controller respectively. The constants k_c , τ_i and τ_d are called the ‘proportional gain’, the ‘integral time’ and the ‘derivative time’ respectively. Sometimes, the term proportional band (PB) is used instead of the proportional gain of k_c . The PB is defined as $PB = (u_{\max} - u_{\min})/k_c$, where, u_{\max} and u_{\min} are the upper limit and the lower limit of the control output (equivalently, the actuator output). For example, $PB = 100/k_c$ in the case of $u_{\max} - u_{\min} = 100$. So, the proportional gain k_c can be calculated simply if the PB is given.

As shown in (4.4), the PID controller is just a simple function of which the input is $y_s(t) - y(t)$ and the output is $u(t)$. It has been most widely used in industry and it is famous for its simplicity as well as the excellent control performance and robustness. The PID controller has the three tuning parameters k_c , τ_i and τ_d , which should be set appropriately with in-depth consideration of the process dynamics.

The setpoint $y_s(t)$ and the parameters k_c , τ_i and τ_d are set by the user. The process output $y(t)$ is measured. Then, it is straightforward to calculate the output of the PID controller. The outputs of the integral part $u_I(t)$ and the derivative part $u_D(t)$ are usually calculated by the numerical integration method and the numerical derivative method respectively. For detailed descriptions on the numerical integration and the numerical derivative, refer to Chapter 2.

Example 4.1

Calculate the output ($u(t)$ for $t \geq 0$) of the PID controller if $y_s(t) = 1$ for $t \geq 0$ and $y(t) = 0$ for $t \geq 0$. The parameters are $k_c = 1.0$, $\tau_i = 5.0$ and $\tau_d = 0.2$.

Solution $u_P(t) = 1.0(1.0 - 0.0) = 1.0$ for $t \geq 0$, $u_I(t) = 1.0 \int_0^t (1.0 - 0.0) dt/5.0 = t/5.0$ for $t \geq 0$ and $u_D(t) = 0.2d(1.0 - 0.0)/dt = 0.0$ for $t \geq 0$. So, $u(t) = 1.0 + t/5.0$.

Example 4.2

Calculate the output ($u(t)$ for $t \geq 0$) of the PID controller if $y_s(t) = 1$ for $t \geq 0$, $y_s(t) = 0$ for $t < 0$ and $y(t) = 0$. The parameters are $k_c = 1.0$, $\tau_i = 5.0$ and $\tau_d = 0.2$.

Solution $u_P(t) = 1.0(1.0 - 0.0) = 1.0$ for $t \geq 0$, $u_I(t) = 1.0 \int_0^t (1.0 - 0.0) dt/5.0 = t/5.0$ for $t \geq 0$, $u_D(t) = 0.2d(1.0 - 0.0)/dt = 0.0$ for $t > 0$, $u_D(t) = \infty$ for $t = 0$. So, $u(t) = 1.0 + t/5.0$ for $t > 0$ and $u(t) = \infty$ for $t = 0$.

Example 4.3

Calculate the output ($u(t)$ for $t \geq 0$) of the PID controller if $y_s(t) = 1$ for $t \geq 0$, $y_s(t) = 0$ for $t < 0$ and $y(t) = 1 - \exp(-t)$ for $t \geq 0$, $y(t) = 0$ for $t < 0$. The parameters are $k_c = 1.0$, $\tau_i = 5.0$ and $\tau_d = 0.2$.

Solution $u_p(t) = \exp(-t)$ for $t \geq 0$, $u_i(t) = \int_0^t \exp(-\tau) d\tau / 5.0 = [1 - \exp(-t)] / 5.0$ for $t \geq 0$, $u_d(t) = -0.2\exp(-t)$ for $t > 0$, $u_d(t) = \infty$ for $t = 0$. So, $u(t) = 0.6\exp(-t) + 0.2$ for $t > 0$ and $u(t) = \infty$ for $t = 0$.

Example 4.4

What is the transfer function of the PID controller (4.4)?

Solution The input and the output of the PID controller are $y_s(t) - y(t)$ and $u(t)$ respectively. Then, the transfer function is

$$G_c(s) = \frac{u(s)}{y_s(s) - y(s)} = k_c + \frac{k_c}{\tau_i s} + k_c \tau_d s$$

4.1.2 Implementation of Proportional–Integral–Derivative Controllers in Computers/Microprocessors

Use the following three steps to implement the algorithm of the PID controller in computers or microprocessors. First, read the process output from the sensor. Second, calculate the control output of the PID controller. Third, send out the control output to the actuator. In the second step, the integral part and the derivative part can be calculated by a numerical integration method and a numerical derivative method respectively. Refer to the following steps for the detailed descriptions on the implementation of the PID controller. The Euler method and the backward difference method are used for the integral part and the derivative part respectively.

1. Read the present (k th sampling) process output $y(k)$ from the sensor.
2. Calculate the controller output on the basis of the present and one-step-before data.

$$u_p(k) = k_c(y_s(k) - y(k)) \quad \text{proportional part} \quad (4.5)$$

$$u_i(k) = u_i(k-1) + \frac{k_c}{\tau_i} (y_s(k) - y(k)) \Delta t \quad \text{integral part} \quad (4.6)$$

$$u_d(k) = k_c \tau_d \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.7)$$

$$u(k) = u_p(k) + u_i(k) + u_d(k) + u_{\text{bias}} \quad (4.8)$$

3. Send the controller output to the actuator. When the time passes as much as the sampling time Δt , repeat from step 1 with the $k + 1$ -th sampling.

Note that the bias term of u_{bias} in (4.8) is usually used to set the reference value of the control output. Before you start or restart the PID controller, you should set the bias term to the present

control output. Otherwise, a big perturbation at the beginning of the PID control may swing the process for quite a while.

Example 4.5

Simulate the following third-order plus time-delay process controlled by a PID controller using the Euler method with $\Delta t = 0.01$. In this case, $u_{\text{bias}} = 0$ because $u(t) = 0$ for $t < 0$.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = -0.3 \frac{du(t-0.2)}{dt} + u(t-0.2) \quad (4.9)$$

$$u(t) = 1.5(y_s(t) - y(t)) + \frac{1.5}{3.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau + 1.5 \times 0.5 \frac{d(y_s(t) - y(t))}{dt} \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (4.10)$$

$$y_s(t) = 1.0 \quad \text{for } t \geq 1, \quad y_s(t) = 0.0 \quad \text{for } t < 1 \quad (4.11)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (4.12)$$

Solution To solve the high-order differential equation, it should be rewritten to the following state-space differential equation according to (1.175)–(1.183). Then, it is straightforward to solve the state-space system.

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t-0.2) \quad (4.13)$$

$$y(t) = Cx(t) \quad (4.14)$$

$$x(0) = [0 \quad 0 \quad 0]^T \quad (4.15)$$

$$A = \begin{bmatrix} 0 & 0 & -1 \\ 1 & 0 & -3 \\ 0 & 1 & -3 \end{bmatrix} \quad (4.16)$$

$$B = [1 \quad -0.3 \quad 0]^T \quad (4.17)$$

$$C = [0 \quad 0 \quad 1] \quad (4.18)$$

The MATLAB code to simulate the closed-loop control system (4.9)–(4.12) and the simulation results are given in Table 4.1 and Figure 4.1 respectively. Here, s in the code corresponds to u_i . The historical data $h_u(j) = 0$, $j = 1, 2, \dots, 1000$, are initially filled with

Table 4.1 MATLAB code to simulate the closed-loop control system of Example 4.5.

pid_ex1.m	g_pid_ex1.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/ delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_ex1(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)','y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_ex1(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre> <hr/> <pre> command window >> pid_ex1 </pre>

zero because $u(t)=0$ for $t<0$. In the case that u_{bias} is set to a nonzero value, this can be incorporated by setting the initial value of the integral part. For example, set $s = u_{\text{bias}}$ instead of $s=0$ in Table 4.1 to incorporate $u_{\text{bias}} \neq 0$. In this example, $s=0$ because of $u_{\text{bias}}=0$.

Example 4.6

Simulate the third-order plus time-delay process (4.19) controlled by a PID controller using the Euler method with $\Delta t = 0.01$. This example is the same as Example 4.5 except that the limits of the control output are enforced.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = -0.3 \frac{du(t-0.2)}{dt} + u(t-0.2) \quad (4.19)$$

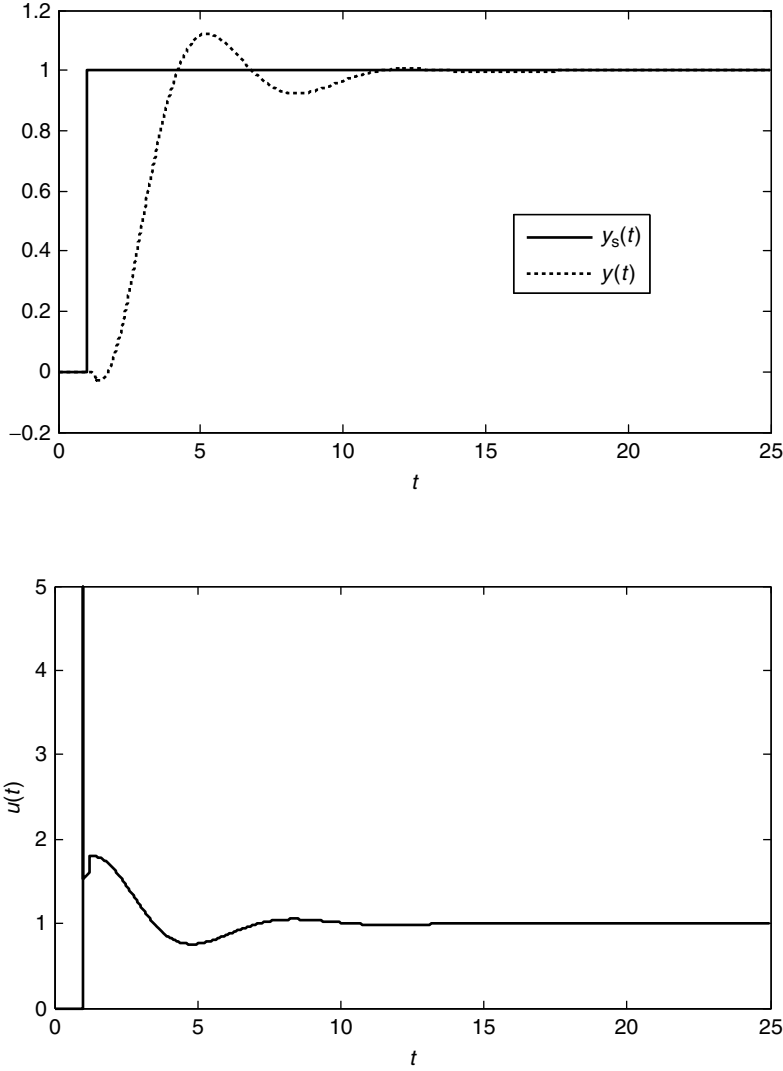


Figure 4.1 Simulation result of Table 4.1 in the case of $u_{\text{bias}}=0$.

$$\begin{aligned}
 u_{\text{PID}}(t) &= 1.5(y_s(t) - y(t)) + \frac{1.5}{3.0} \int_0^t (y_s(\tau) - y(\tau)) \, d\tau + 1.5 \\
 &\times 0.5 \frac{d(y_s(t) - y(t))}{dt} \quad \text{for } t \geq 0 \quad \text{and} \quad u_{\text{PID}}(t) = 0 \quad \text{for } t < 0
 \end{aligned}
 \tag{4.20}$$

$$\begin{aligned}
 u(t) &= u_{\text{PID}}(t) \quad \text{for } |u_{\text{PID}}(t)| \leq 1.2, \\
 u(t) &= 1.2 \quad \text{for } u_{\text{PID}}(t) > 1.2, \\
 u(t) &= -1.2 \quad \text{for } u_{\text{PID}}(t) < -1.2
 \end{aligned}
 \tag{4.21}$$

$$y_s(t) = 1.0 \quad \text{for } t \geq 1, \quad y_s(t) = 0.0 \quad \text{for } t < 1 \quad (4.22)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (4.23)$$

Solution The MATLAB code to simulate the closed-loop control system (4.19)–(4.23) and the simulation results are given in Table 4.2 and Figure 4.2 respectively. We realize that the

Table 4.2 MATLAB code to simulate the closed-loop control system of Example 4.6.

pid_ex2.m	g_pid_ex2.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/ delta_t; if (u>1.2) u=1.2; end if (u<-1.2) u=-1.2; end ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_ex2(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_ex2(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> pid_ex2 </pre>

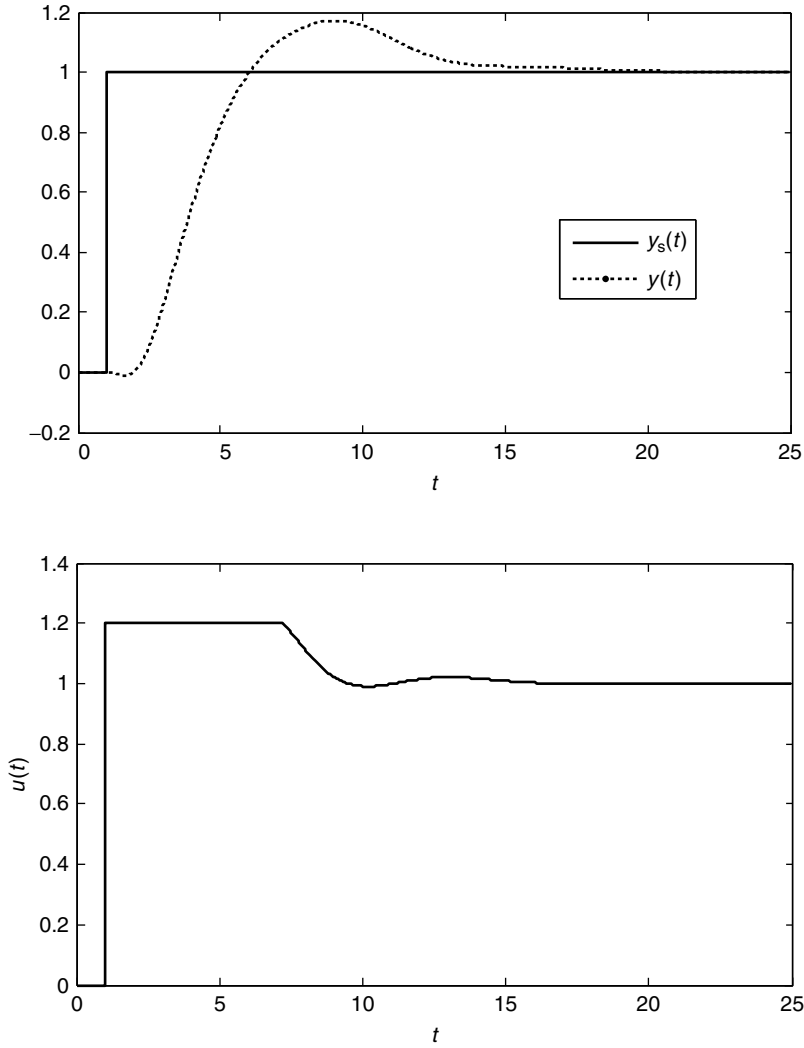


Figure 4.2 Simulation result of Table 4.2.

response of the process becomes sluggish due to the limits of the control output. Also, the process output stays above the setpoint for a long time.

Example 4.7

Simulate Example 4.5 again with a sampling time $\Delta t_s = 0.5$ for the PID controller.

Solution The MATLAB code to simulate the process controlled by a PID controller for which the sampling time is $\Delta t_s = 0.5$ and the simulation results are given in Table 4.3

Table 4.3 MATLAB code to simulate the closed-loop control system of Example 4.7.

<pre>pid_ex3.m clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); % delta_t for Euler delta_ts=0.5; t_previous=-delta_ts; %sampling time for PID C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; % sample the data for PID every delta_ts if (t>=(t_previous+delta_ts-delta_t*0.5)) if (t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_ts; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/ delta_ts; ysb=ys; yb=y; t_previous=t;% one sampling before end u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_ex3(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array);</pre>	<pre>g_pid_ex3.m function [dx_dt]2=g_pid_ex3(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end command window >> pid_ex3</pre>
---	--

and Figure 4.3 respectively. Note that the time interval Δt for the Euler method should be small enough to solve the differential equation with acceptable accuracy. So, the code of Example 4.5 should be modified as shown in Table 4.3. delta_t*0.5 in the code is just to compensate for the round-off error of the computer. Comparing Figure 4.3 with Figure 4.1,

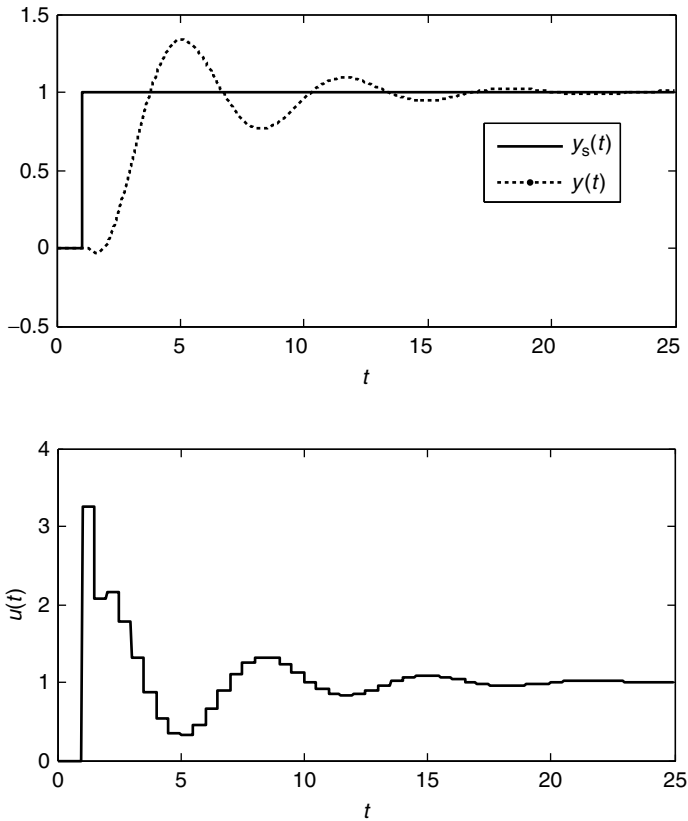


Figure 4.3 Simulation result of Table 4.3.

the larger sampling time can degrade the control performance compared with the case of the smaller sampling time because the PID controller of the larger sampling time responds more infrequently.

Example 4.8

Simulate Example 4.5 again when the process output is contaminated by uniformly distributed random noises between -0.1 and 0.1 . The sampling time of the PID controller is 0.01 .

Solution The MATLAB code to simulate the case of the measurement noises and the simulation results are given in Table 4.4 and Figure 4.4 respectively. It should be noted that the control output of the PID controller fluctuates severely for the measurement noises. This can cause damage to the actuator. The severe fluctuation originates from the derivative action of the PID controller. As shown in (4.7), it is clear that the derivative action is extremely sensitive to the measurement noises if the sampling time for the PID

Table 4.4 MATLAB code to simulate the closed-loop control system of Example 4.8.

<pre>pid_ex4.m clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; rand('seed',0); noise=0.2*(rand(1,n)-0.5); for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_ex4(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x+noise(i); t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)','y(t)'); figure(2); plot(t_array,u_array);</pre>	<pre>g_pid_ex4.m function [dx_dt]=g_pid_ex4(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end command window >> pid_ex4</pre>
---	---

controller is very small. So, too small a sampling time for the PID controller is not recommended.

Example 4.9

Simulate Example 4.8 again with a sampling time of 0.1 for the PID controller.

Solution The MATLAB code to simulate the case of the measurement noises with a sampling time of 0.1 and the simulation results are given in Table 4.5 and Figure 4.5 respectively. The PID

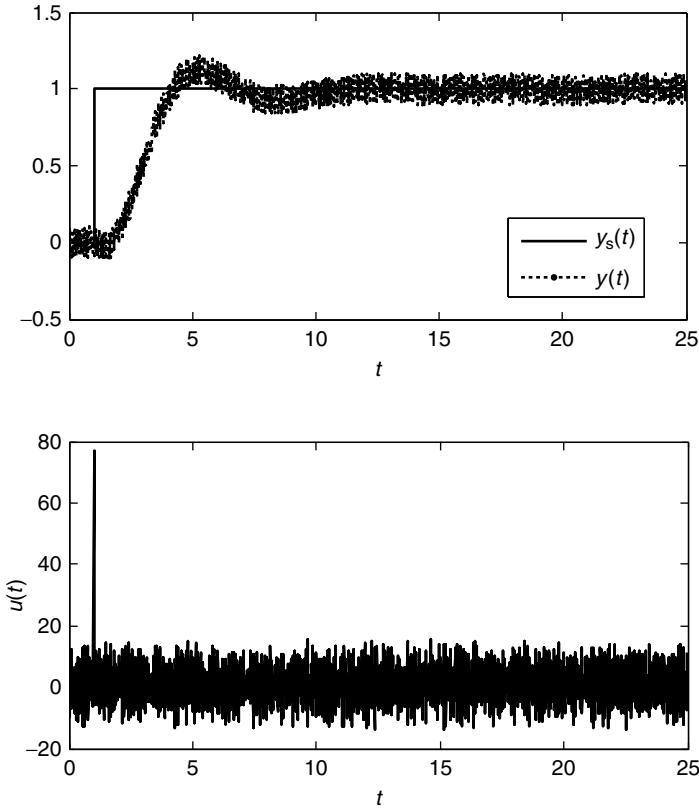


Figure 4.4 Simulation result of Table 4.4.

controller shows an acceptable fluctuation of the control output for a sampling time of 0.1 and almost the same control performance as in the case of the very small sampling time of Figure 4.4.

4.2 Roles of Three Parts of Proportional–Integral–Derivative Controllers

The control output $u_P(t)$ of the proportional part is proportional to the error $y_s(t) - y(t)$ as shown in (4.1). This means that the proportional part plays a role in pushing the process output to the setpoint as much as the error.

For the usual processes (open-loop stable processes), the control output should be a nonzero constant to keep the process output to a nonzero setpoint. For example, imagine a control system to control the room temperature $y(t)$ of a house in winter by adjusting the fuel consumption $u(t)$ of the boiler. It is obvious that a constant amount of the fuel $u(t)$ should be provided continuously to keep up an appropriate room temperature ($y_s = y(t) = \text{a nonzero setpoint}$). Meanwhile, the following proportional–derivative (PD) controller output is $u(t) = 0$

Table 4.5 MATLAB code to simulate the closed-loop control system of Example 4.9.

pid_ex4.m	g_pid_ex4.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t);% delta_t for Euler delta_ts=0.1; t_previous=-delta_ts; % sampling time for PID C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; rand('seed',0); noise=0.2*(rand(1,n)-0.5); for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; % sampling for PID if (t>=(t_previous+delta_ts-delta_t*0.5)) if (t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_ts; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/ delta_ts; ysb=ys; yb=y; % one sampling before t_previous=t; end u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_ex5(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x+noise(i); t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_ex5(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre> <hr/> <p>command window</p> <pre> >> pid_ex4 </pre>

if the error $y_s - y(t)$ is zero at steady state:

$$u(t) = u_P(t) + u_D(t) = k_c(y_s - y(t)) + k_c\tau_d \frac{d(y_s - y(t))}{dt} \quad (4.24)$$

That is, the output of the PD controller cannot be a nonzero constant when the error is zero at steady state. So, the PD controller cannot keep the process output to a nonzero setpoint for

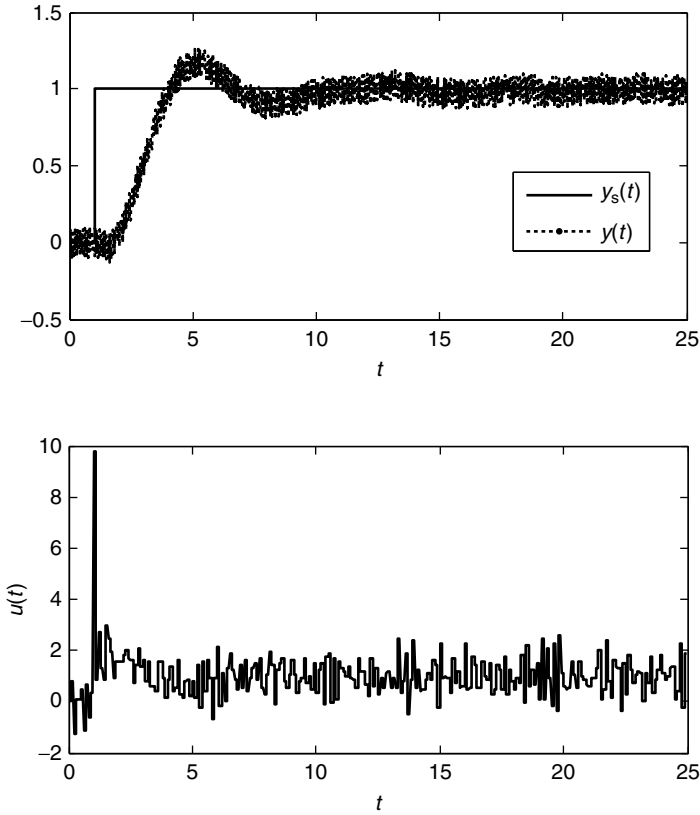


Figure 4.5 Simulation result of Table 4.5.

open-loop stable processes, resulting in an offset. The offset is defined as the error $y_s(t) - y(t)$ at steady state.

The offset can be calculated easily. Equation (4.25) represents the relationship between the process output and the controller output at steady state, where k is called the static gain or DC gain of the process. Equation (4.24) becomes (4.26) at steady state.

$$y_{ss}(t) = k u_{ss}(t) \quad (4.25)$$

$$u_{ss}(t) = k_c (y_s - y_{ss}(t)) \quad (4.26)$$

where the subscript 'ss' denotes steady state. From (4.25) and (4.26), the offset is

$$y_s - y_{ss}(t) = \frac{y_s}{1 + k k_c} \quad (4.27)$$

If the integral part is added to the controller, then the offset can be rejected because the integral part can be a nonzero constant even though the present error is zero. That is, (4.4)

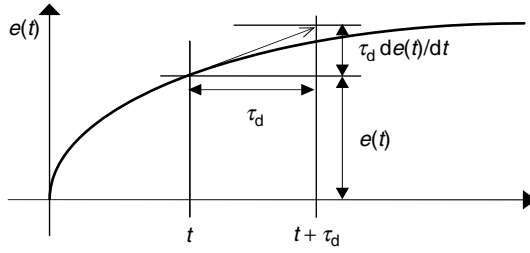


Figure 4.6 Extrapolation using the derivative of the error.

at steady state becomes (4.28) due to the accumulated error if the offset is zero:

$$u_{ss}(t) = \frac{k_c}{\tau_i} \int_0^t (y_s - y(\tau)) d\tau = \text{nonzero constant} \quad (4.28)$$

From comparison of (4.25) and (4.28), the integral of the error at steady state can be calculated:

$$\int_0^\infty (y_s - y(\tau)) d\tau = \frac{\tau_i}{k k_c} y_s \quad (4.29)$$

In summary, the conclusion is that the integral part of the PID controller plays an important role in rejecting the offset. This is possible because the integral term can be a nonzero constant due to the accumulated error.

$\tau_d d(y_s - y(t))/dt$ represents approximately the increment of the error after τ_d from the present time t , as shown in Figure 4.6. Therefore, the derivative part plays a role in rejecting the future error in advance by increasing the control output in proportion to the future error. This means that the derivative part can enhance the robustness of the PID controller by considering the future change of the error.

Example 4.10

Find the final values of $y(t)$ and $u(t)$ of the process (4.30) and the controller (4.31) at steady state. Assume that $y_s(t) = y_{s,ss}$ is constant and the signals $y(t)$ and $u(t)$ converge to constant values.

$$\frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (4.30)$$

$$u(t) = 1.2(y_s(t) - y(t)) + 0.6 \frac{d(y_s(t) - y(t))}{dt} \quad (4.31)$$

Solution At steady state, all the derivatives are zero and the delayed value is the same as the present value (that is, $y_{ss}(t - \theta) = y_{ss}(t)$). Then, (4.30) and (4.31) become

$$y_{ss} = u_{ss}, \quad u_{ss} = 1.2(y_{s,ss} - y_{ss}) \quad (4.32)$$

$$y_{ss} = 1.2 y_{s,ss} / 2.2 \quad (4.33)$$

where the subscript 'ss' denotes steady state. Note that the PD controller shows the offset as shown in (4.33).

Example 4.11

Can the P controller (4.35) reject the offset for the integrating process (4.34) for the constant setpoint of $y_s(t) = y_{s,ss}$? The integrating process means that the process has a zero pole as shown in (4.34):

$$y(s) = \frac{3}{s(s+1)^2} u(s) \quad (4.34)$$

$$u(s) = 2.5(y_s(s) - y(s)) \quad (4.35)$$

Solution Equations (4.34) and (4.35) are equivalent to the following equations:

$$\frac{d^3 y(t)}{dt^3} + 2 \frac{d^2 y(t)}{dt^2} + \frac{dy(t)}{dt} = 3u(t) \quad (4.36)$$

$$u(t) = 2.5(y_s(t) - y(t)) \quad (4.37)$$

The zero offset means $y_{ss}(t) = y_{s,ss}$. Then, $u_{ss} = 0$ from (4.37). The process also satisfies $0 = u_{ss}$ at steady state, which means that a zero control output is needed to keep up a nonzero process output for the integrating process. So, the offset can be rejected by the P controller. Meanwhile, there is always an offset if the P controller is used for an open-loop stable process because a nonzero control output is needed to keep up the nonzero process output for the open-loop stable process.

Example 4.12

Calculate the final value of the integral part of the PID controller (4.38) for the process (4.30) when the offset is zero.

$$u(t) = k_c e(t) + \frac{k_c}{\tau_i} \int_0^t e(\tau) d\tau + k_c \tau_d \frac{de(t)}{dt} \quad (4.38)$$

Solution Because the offset is zero, $e_{ss}(t) = 0$. The derivative $de_{ss}(t)/dt$ is zero at steady state. So, the output of the PID controller (4.38) becomes $u_{ss} = k_c/\tau_i \int_0^t e(\tau) d\tau$ at steady state, which is not necessarily zero due to the accumulation of the past errors even though the present error is zero ($e_{ss}(t) = 0$). And $y_s = u_{ss}$ is valid for (4.30) at steady state. So, $y_s = k_c/\tau_i \int_0^t e(\tau) d\tau$.

Example 4.13

How can PI (or I, PID) controllers reject the offset for the open-loop stable process?

Solution Because the controller contains the integral term, the controller output can be a nonzero value (corresponding to the setpoint) due to the accumulated integral term even though

the present error is zero. Meanwhile, P or PD controllers cannot reject the offset for the open-loop stable process because the control output of the P or PD controllers is zero if the present error is zero.

Example 4.14

Simulate the following SOPTD process controlled by the P controller and confirm that the offset converges to the expected value:

$$\frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} + y(t) = 0.7u(t-0.3) \quad (4.39)$$

$$u(t) = 3.5(y_s(t) - y(t)) \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (4.40)$$

$$y_s(t) = 1.0 \quad \text{for } t \geq 1, \quad y_s(t) = 0.0 \quad \text{for } t < 1 \quad (4.41)$$

$$\left. \frac{d^2y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (4.42)$$

Solution At steady state, all the derivative values are zero and the delayed value is the same as the present value. Then, (4.39) and (4.40) become

$$y_{ss} = 0.7u_{ss}, \quad u_{ss} = 3.5(1 - y_{ss}) \quad (4.43)$$

So, the final value of the process output is $y_{ss} = 0.7101$ and the offset is $1 - y_{ss} = 0.2899$. The MATLAB code and the simulation results of Table 4.6 and Figure 4.7 respectively confirm this.

Example 4.15

Simulate the following SOPTD process controlled by the PID controller and confirm that the integral part of the PID controller converges to the expected value:

$$\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = -0.2\frac{du(t-0.3)}{dt} + 0.5u(t-0.3) \quad (4.44)$$

$$u(t) = 3.0(y_s(t) - y(t)) + \frac{3.0}{3.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau \quad \text{for } t \geq 0, \quad u(t) = 0 \quad \text{for } t < 0 \quad (4.45)$$

$$y_s(t) = 1.0 \quad \text{for } t \geq 1, \quad y_s(t) = 0.0 \quad \text{for } t < 1 \quad (4.46)$$

Table 4.6 MATLAB code to simulate the PID control system of Example 4.14.

pid_role_ex5.m	g_pid_role_ex5.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 1]; theta=0.3; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=3.5; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if (t>1) ys=1.0; else ys=0.0; end u=kc*(ys-y); u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_role_ex5(y,x,h_u (1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_role_ex5 (y,x,u) A=[0 -1; 1 -2]; B=[0.7 0]'; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> pid_role_ex5 </pre>

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (4.47)$$

Solution At steady state, all the derivative values are zero and the delayed value is the same as the present value. Also, the offset is zero because the I term is included. Then, (4.44) and (4.45) become

$$1 = y_{ss} = 0.5u_{ss}, \quad u_{ss} = \frac{3.0}{3.0} \int_0^t (1 - y(\tau)) d\tau \quad (4.48)$$

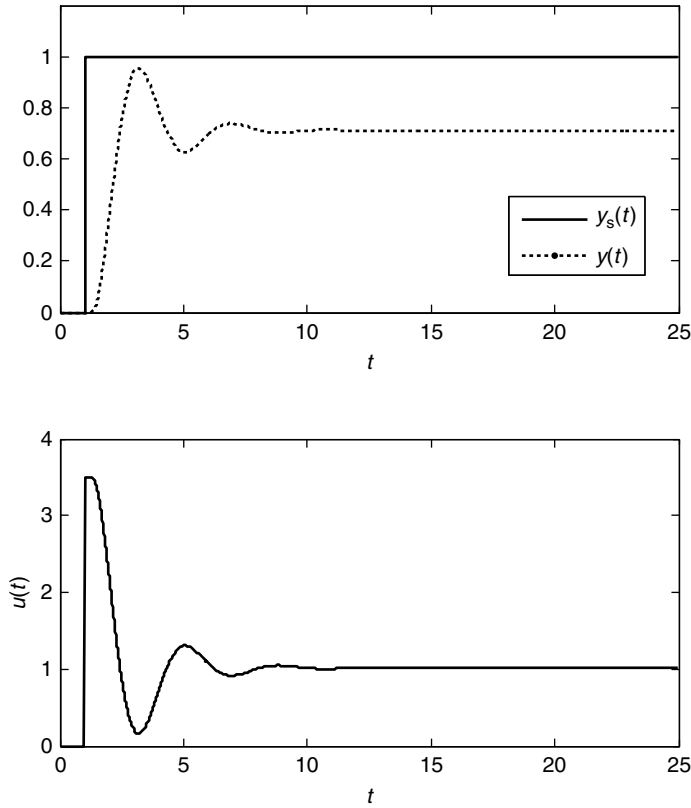


Figure 4.7 Simulation result of Table 4.6.

So, the integral part of $(3.0/3.0) \int_0^t (1 - y(\tau)) \, d\tau$ converges to 2.0. The MATLAB code and the simulation results of Table 4.7 and Figure 4.8 respectively confirm this.

4.3 Integral Windup

Actuators, such as valves, motors, electric powers and so on, always have a lower limit and an upper limit. When the actuators are at their limit values (which is called saturation), the dynamics of the process output become much more sluggish than the case of no actuator limitations for a big step setpoint change or big disturbance. So, the integral part of the PID controller increases rapidly. This phenomenon is called the integral windup. Consider the simulation result in Figure 4.9.

As shown in Figure 4.9, the control output (actuator) is saturated from $t = 1.0$ to $t = 9.0$. During the period from $t = 1.0$ to $t = 5.5$, the integral part of the PID controller increases rapidly because the error decreases slowly due to the saturation. Note that the final integral part of the PID controller is already determined (1.0 in Figure 4.9). Owing to the actuator saturation, the integral part up to the rising time would be much bigger than the final value, as

Table 4.7 MATLAB code to simulate the PID control system of Example 4.15.

pid_role_ex6.m	g_pid_role_ex6.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01;n=round(t_final/delta_t); C=[0 1]; theta=0.3; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=3.0; ti=3.0; td=0.0; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; I_array(i)=s; if (t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_role_ex6(y,x,h_u (1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array,t_array,I_array); legend('u(t)', 'u_{i}(t)'); </pre>	<pre> function [dx_dt]=g_pid_role_ex6(y,x,u) A=[0 -1; 1 -3]; B=[0.5 -0.2]'; dx_dt=A*x+B*u; end </pre> <hr/> <p>command window</p> <pre> >> pid_role_ex6 </pre>

shown in Figure 4.9. So, a big overshoot (i.e. a big negative error) is inevitable to reduce the already accumulated integral part to the final integral part (refer to the integral part from $t = 5.5$ to $t = 15.0$ in Figure 4.9). In this way, the integral windup degrades the control performance.

4.3.1 Anti-Windup

Several techniques are available to prevent integral windup. The two anti-windup methods of the conditional integration and back calculation are introduced.

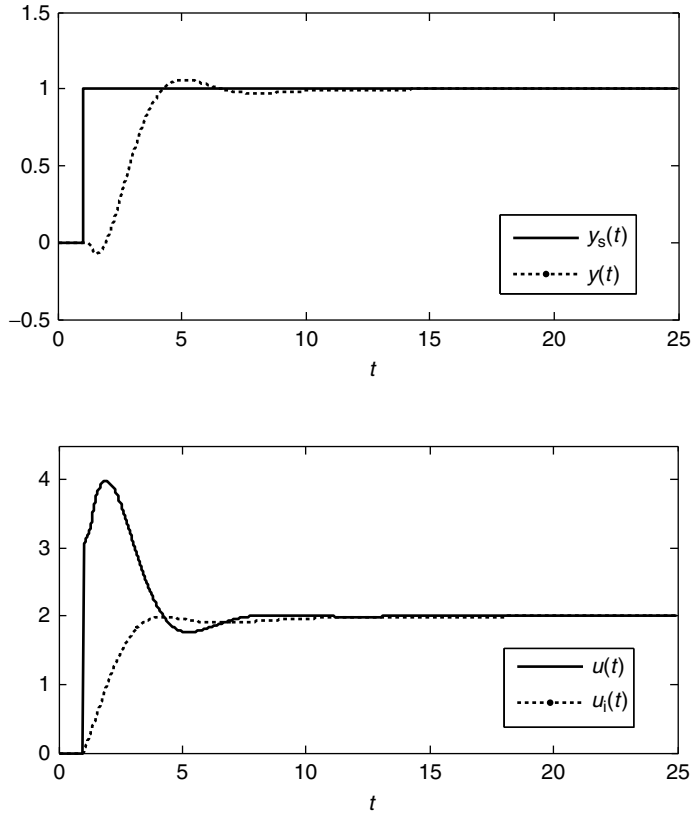


Figure 4.8 Simulation result of Table 4.7.

4.3.1.1 Conditional Integration

The simplest anti-windup method to prevent integral windup is freezing the integral action when the actuator is at the limit value. The code for the anti-windup method is exemplified as follows.

$$\text{If } u(k) > u_{\max}, \text{ no update of the integral part like } u_i(k) = u_i(k-1) \text{ and set } u(k) = u_{\max} \quad (4.49)$$

$$\text{If } u(k) < u_{\min}, \text{ no update of the integral part like } u_i(k) = u_i(k-1) \text{ and set } u(k) = u_{\min} \quad (4.50)$$

where u_{\min} and u_{\max} are the lower limit and the upper limit respectively. Then, the calculation of the PID control with the anti-windup technique for the k th sample is summarized as follows:

$$u_p(k) = k_c(y_s(k) - y(k)) \quad \text{proportional part} \quad (4.51)$$

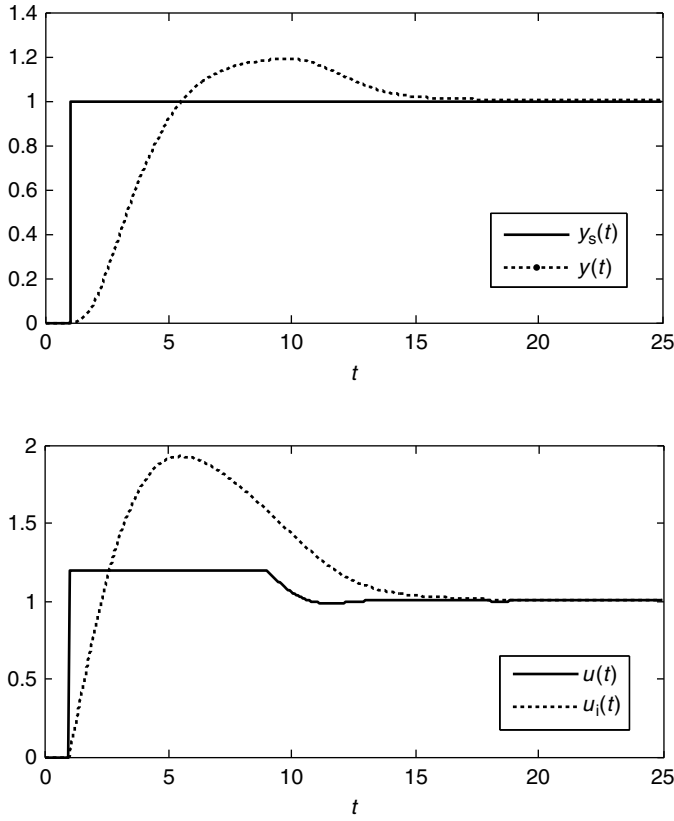


Figure 4.9 Typical closed-loop responses of a PID control system with actuator saturation.

$$u_i(k) = u_i(k-1) + \frac{k_c}{\tau_i} (y_s(k) - y(k)) \Delta t \quad \text{integral part} \quad (4.52)$$

$$u_d(k) = k_c \tau_d \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.53)$$

$$u(k) = u_p(k) + u_i(k) + u_d(k) \quad (4.54)$$

$$\text{If } u(k) > u_{\max}, \quad u_i(k) = u_i(k-1) \text{ (no update) and } u(k) = u_{\max} \quad (4.55)$$

$$\text{If } u(k) < u_{\min}, \quad u_i(k) = u_i(k-1) \text{ (no update) and } u(k) = u_{\min} \quad (4.56)$$

Other conditional integrations can also be used. For example, the integral action can be stopped when the control output or the integral part is larger than a designated value.

Table 4.8 MATLAB code to simulate a closed-loop control system with an anti-windup technique of the conditional integration.

<pre>pid_antiwindup1.m clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.0; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=2.0; ti=2.5; td=0.5; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; I_array(i)=s; if (t>1) ys=1.0; else ys=0.0; end s_n=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s_n+kc*td*((ys-y)-(ysb-ysb))/delta_t; if ((u<=1.2) & (u>=-1.2)) s=s_n; end if (u>1.2) u=1.2; end if (u<-1.2) u=-1.2; end ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_antiwindup1 (y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array,t_array,I_array); legend('u(t)', 'u_{i}(t)');</pre>	<pre>g_pid_antiwindup1.m function [dx_dt]=g_pid_antiwindup1 (y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 0 0.0]'; dx_dt=A*x+B*u; end command window >> pid_antiwindup1</pre>
---	---

Consider the simulation in Table 4.8 and Figure 4.10. Equation (4.57) is controlled by the PID controller $k_c = 2.0$, $\tau_i = 2.5$ and $\tau_d = 0.5$ and the anti-windup technique (4.49)–(4.56) is used. The lower limit and the upper limit of the actuator are -1.2 and 1.2

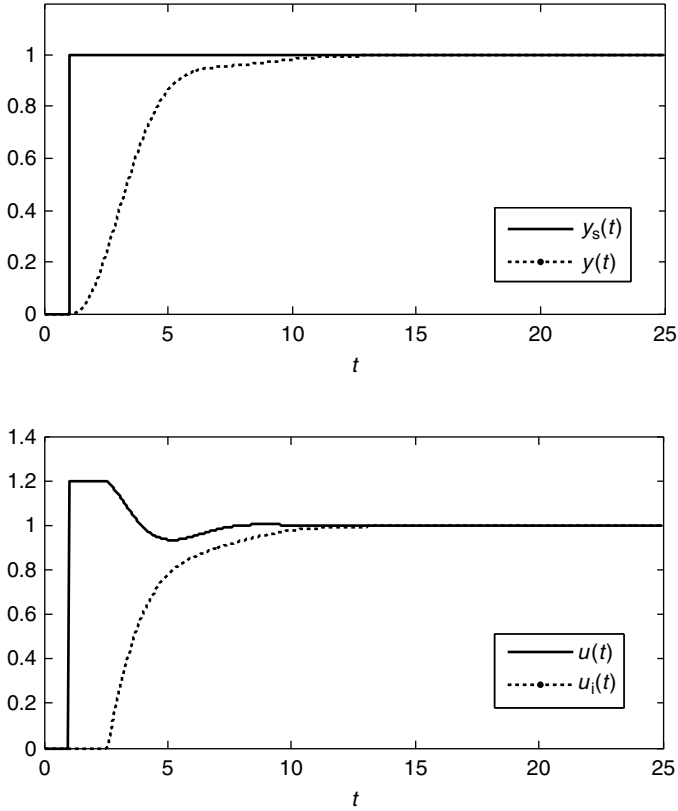


Figure 4.10 Simulation result of Table 4.8.

respectively.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = u(t) \quad (4.57)$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad \left. \frac{du(t)}{dt} \right|_{t=0} = u(0) = 0 \quad (4.58)$$

The MATLAB code to simulate the closed-loop control system with the anti-windup technique and the simulation results are given in Table 4.8 and Figure 4.10 respectively.

4.3.1.2 Back Calculation

The other anti-windup technique uses an internal feedback loop, which makes the integral part converge to the limit of the actuator. The calculation of the PID control with the anti-windup technique of the back calculation for the k th sample is summarized as follows:

$$u_p(k) = k_c(y_s(k) - y(k)) \quad \text{proportional part} \quad (4.59)$$

$$u_d(k) = k_c \tau_d \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.60)$$

$$u_{\text{PID}}(k) = u_p(k) + u_i(k) + u_d(k) \quad (4.61)$$

$$\text{If } u_{\text{PID}}(k) > u_{\max}, \quad u(k) = u_{\max} \quad (4.62)$$

$$\text{If } u_{\text{PID}}(k) < u_{\min}, \quad u(k) = u_{\min} \quad (4.63)$$

$$\text{If } u_{\text{PID}}(k) \geq u_{\min} \quad \text{and} \quad u_{\text{PID}}(k) \leq u_{\max}, \quad u(k) = u_{\text{PID}}(k) \quad (4.64)$$

$$u_i(k+1) = u_i(k) + \left[\frac{1}{\tau_i} (u(k) - u_{\text{PID}}(k)) + \frac{k_c}{\tau_i} (y_p(k) - y(k)) \right] \Delta t \quad \text{integral part} \quad (4.65)$$

In (4.65), the internal feedback term $(u(k) - u_{\text{PID}}(k))/\tau_i$ is zero if the actuator is not saturated as shown in (4.64). If the actuator is saturated at the upper limit like (4.62), the term decreases the integral part $u_i(k+1)$ because the term is negative. On the other hand, it increases the integral part if the actuator is saturated at the lower limit like (4.63) because the term is positive. So, the term $(u(k) - u_{\text{PID}}(k))/\tau_i$ in (4.65) makes the output of the PID controller converge to the operation range of the actuator. τ_i is called the tracking time constant. The recommended range of the tracking time constant is $\tau_d \leq \tau_i \leq \tau_p$.

Consider the simulation in Table 4.9 and Figure 4.11. This is the same as in Table 4.8 except that the anti-windup technique used is the back calculation.

4.4 Commercial Proportional–Integral–Derivative Controllers

Various modifications of the PID controller are introduced in this section, followed by unified structures of commercial PID controllers. Before tuning the parameters of the commercial PID controller, one must check the structure of the PID controller because its structure can be totally different from the structure of the ideal PID controller of (4.4).

4.4.1 Noninteracting, Interacting and Parallel Proportional–Integral–Derivative Controllers

The structure of a noninteracting PID controller is the same as that of the ideal PID controller in (4.4) as follows:

$$G_c(s) = \frac{u(s)}{y_s(s) - y(s)} = k_c + \frac{k_c}{\tau_i s} + k_c \tau_d s \quad (4.66)$$

The implementation form of the noninteracting PID controller is demonstrated below.

$$u_p(k) = k_c (y_s(k) - y(k)) \quad \text{proportional part} \quad (4.67)$$

$$u_i(k) = u_i(k-1) + \frac{k_c}{\tau_i} (y_s(k) - y(k)) \Delta t \quad \text{integral part} \quad (4.68)$$

Table 4.9 MATLAB code to simulate a closed-loop control system with an anti-windup technique of the back calculation.

pid_antiwindup2.m	g_pid_antiwindup2.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.0; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=2.0; ti=2.5; td=0.5; tt=2.0; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; I_array(i)=s; if (t>1) ys=1.0; else ys=0.0; end u_pid=kc*(ys-y)+s+kc*td* ((ys-y)-(ysb-yb))/delta_t; if ((u_pid<1.2) & (u_pid>=-1.2)) u=u_pid; end if (u_pid>1.2) u=1.2; end if (u_pid<-1.2) u=-1.2; end s=s+((u-u_pid)/tt+(kc/ti)* (ys-y))*delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_antiwindup2 (y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array,t_array,I_array); legend('u(t)', 'u_{i}(t)');</pre>	<pre> function [dx_dt]=g_pid_antiwindup2 (y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 0 0]'; dx_dt=A*x+B*u; end command window >> pid_antiwindup2</pre>

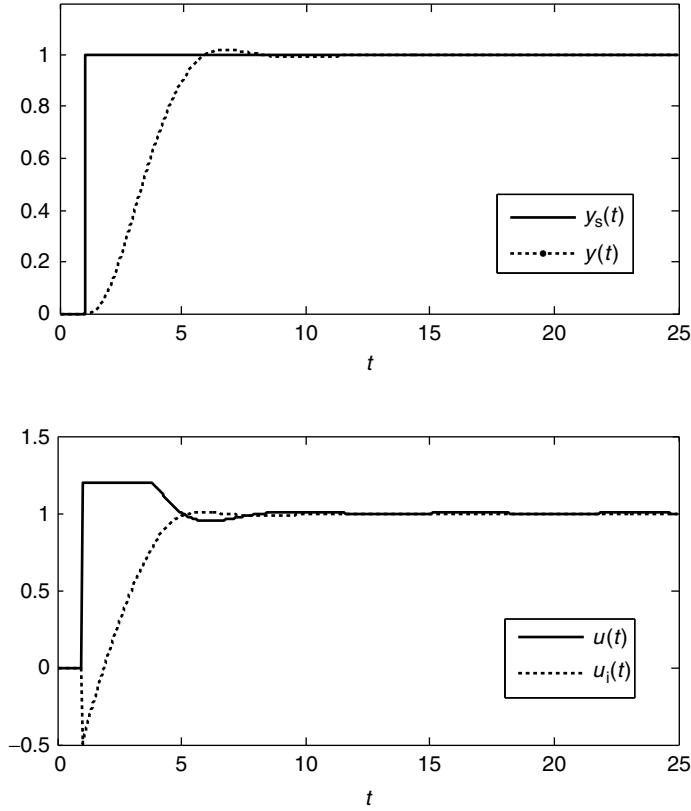


Figure 4.11 Simulation result of Table 4.9.

$$u_d(k) = k_c \tau_d \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.69)$$

$$u(k) = u_p(k) + u_i(k) + u_d(k) \quad (4.70)$$

On the other hand, the transfer function and the implementation form of the interacting PID controller are as follows:

$$G_c(s) = \frac{u(s)}{y_s(s) - y(s)} = k_c^i \left(1 + \frac{1}{\tau_i^i s} \right) (1 + \tau_d^i s) \quad (4.71)$$

$$u_d^i(k) = (y_s(k) - y(k)) + \tau_d^i \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.72)$$

$$u_p^i(k) = k_c^i u_d^i(k) \quad \text{proportional part} \quad (4.73)$$

$$u_i^i(k) = u_i^i(k-1) + \frac{k_c^i}{\tau_i^i} u_d^i(k) \Delta t \quad \text{integral part} \quad (4.74)$$

$$u(k) = u_p^i(k) + u_i^i(k) \quad (4.75)$$

The structure of the parallel PID controller is the same as that of the ideal PID controller in (4.4), but the definitions of the parameters are different, as shown in (4.76):

$$G_c(s) = \frac{u(s)}{y_s(s) - y(s)} = k_c^p + \frac{k_i^p}{s} + k_d^p s \quad (4.76)$$

4.4.2 Relationship between Different Forms of Proportional–Integral–Derivative Controllers

The noninteracting form (4.66) is obtained from the interacting form (4.71) by setting the parameters as follows. The derivation is straightforward.

$$k_c = k_c^i \left(1 + \frac{\tau_d^i}{\tau_i^i} \right) \quad (4.77)$$

$$\tau_i = \tau_i^i + \tau_d^i \quad (4.78)$$

$$\tau_d = \frac{\tau_i^i \tau_d^i}{\tau_i^i + \tau_d^i} \quad (4.79)$$

The interacting form (4.71) is obtained from the noninteracting form (4.66) by setting the parameters as follows. The derivation is also straightforward.

$$k_c^i = \frac{k_c + k_c \sqrt{1 - 4\tau_d/\tau_i}}{2} \quad (4.80)$$

$$\tau_i^i = \frac{\tau_i + \tau_i \sqrt{1 - 4\tau_d/\tau_i}}{2} \quad (4.81)$$

$$\tau_d^i = \frac{\tau_i - \tau_i \sqrt{1 - 4\tau_d/\tau_i}}{2} \quad (4.82)$$

Equations (4.80)–(4.82) are valid only if $\tau_i \geq 4\tau_d$. So, the noninteracting form is more general than the interacting form. The two forms of the parallel and noninteracting forms are interchangeable with the following relationship:

$$k_c^p = k_c, \quad k_i^p = \frac{k_c}{\tau_i}, \quad k_d^p = k_c \tau_d \quad (4.83)$$

Example 4.16

Obtain the interacting PID controller having the same closed-loop response as in Figure 4.1. Confirm it with simulation.

Solution Because the parameters of the noninteracting PID controller are $k_c = 1.5$, $\tau_i = 3.0$ and $\tau_d = 0.5$, the equivalent parameters of the interacting PID controller are $k_c^i = 1.1830$, $\tau_i^i = 2.3660$

Table 4.10 MATLAB code to simulate a closed-loop control system with an interacting PID controller.

pid_interacting1.m	g_pid_interacting1.m
<pre>clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.1830; ti=2.3660; td=0.6340; s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if (t>1) ys=1.0; else ys=0.0; end ud=(ys-y)+td*((ys-y) -(ysb-yb))/delta_t; s=s+(kc/ti)*ud*delta_t; u=kc*ud+s; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_interacting 1(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array, t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array);</pre>	<pre>function [dx_dt]=g_pid_interacting1(y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end</pre>
	<pre>command window >> pid_interacting1</pre>

$\tau_d^i = 0.6340$ by (4.80)–(4.82). The MATLAB code to simulate the closed-loop control system with the interacting PID controller and the simulation results are given in Table 4.10 and Figure 4.12. As shown in Figure 4.12 and Figure 4.1, the same closed-loop response is obtained.

Example 4.17

Obtain the parameters of the parallel PID controller corresponding to the noninteracting PID controller $k_c = 1.5$, $\tau_i = 3.0$ and $\tau_d = 0.5$.

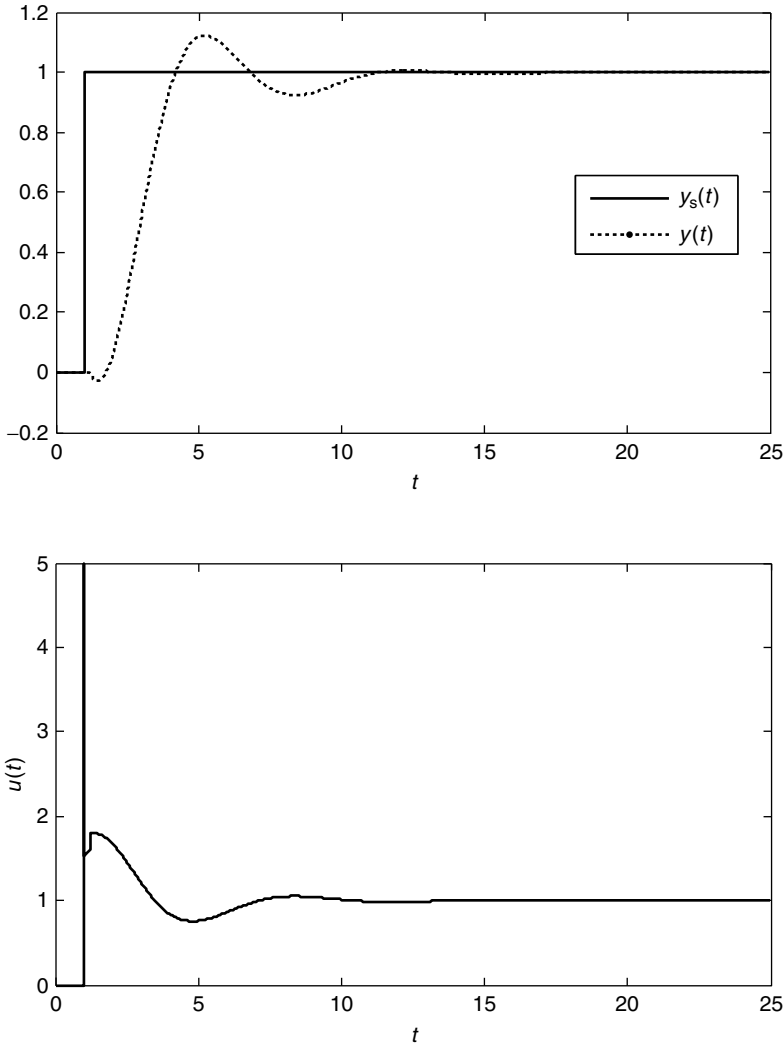


Figure 4.12 Simulation result of Table 4.10.

Solution The equivalent parameters of the parallel PID controller are $k_c^p = 1.5$, $k_i^p = 1.5/3.0$ and $k_d^p = 1.5 \times 0.5$ by (4.83).

4.4.3 Two-Degree-of-Freedom Proportional–Integral–Derivative Controllers

The two-degree-of-freedom PID controller uses the following setpoint weighting:

$$u_p(k) = k_c(w_p y_s(k) - y(k)) \quad \text{proportional part} \quad (4.84)$$

$$u_i(k) = u_i(k-1) + \frac{k_c}{\tau_i} (y_s(k) - y(k))\Delta t \quad \text{integral part} \quad (4.85)$$

$$u_d(k) = k_c \tau_d \frac{(w_d y_s(k) - y(k)) - (w_d y_s(k-1) - y(k-1))}{\Delta t} \quad \text{derivative part} \quad (4.86)$$

$$u(k) = u_p(k) + u_i(k) + u_d(k) \quad (4.87)$$

Usually, w_p ranges from 0 to 1 and w_d is 0 or 1. w_p can reduce the overshoot for a step setpoint change without changing the parameters (k_c , τ_i and τ_d) of the PID controller. This makes it possible to improve the control performances for the setpoint change without degrading the control performances for the input disturbance rejection. The detailed descriptions will be given later in this book. w_d can prevent the derivative kick, which happens when a step setpoint change enters. Consider the simulation results of Figures 4.1, 4.3, 4.4 and 4.5. Large peaks appear at the instant of the step setpoint change, called derivative kick. From (4.86) with $w_d = 1$, it is clear that the derivative kick happens at the instant and it becomes severe as decreasing the sampling time. The derivative kick can cause damage to the actuator. The choice of $w_d = 0$ can remove the derivative kick, called anti-derivative-kick. The control action with $w_d = 1$ is the same as the control action with $w_d = 0$ except at the instant of the step setpoint change, because the setpoint is constant except at the instant.

Example 4.18

Simulate again Figure 4.1 with the anti-derivative-kick ($w_d = 0$). Compare the closed-loop responses.

Solution The MATLAB code to simulate the closed-loop control system with the anti-derivative-kick and the simulation results are given in Table 4.11 and Figure 4.13. As shown in Figure 4.1 and Figure 4.13, the closed-loop response of the anti-derivative-kick is a little bit worse than that of the ideal PID controller. Instead, the derivative kick is removed.

Example 4.19

Simulate again Figure 4.3 with $w_p = 0.6$. Compare the closed-loop responses.

Solution The MATLAB code to simulate the two-degree-of-freedom PID controller and the simulation results are given in Table 4.12 and Figure 4.14. Comparing Figure 4.3 with Figure 4.14, the control performance for the setpoint change is improved without changing the parameters (k_c , τ_i and τ_d) of the PID controller.

4.4.4 Noise-Suppressing Proportional–Integral–Derivative Controllers

The derivative part of the PID controller is very sensitive to measurement noise when the sampling time is small, as shown in Figure 4.4. To overcome this problem, the following

Table 4.11 MATLAB code to simulate a closed-loop control system with an anti-derivative-kick PID controller.

pid_antiderivative1.m	g_pid_antiderivative1.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; wd=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if (t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td* ((wd*ys-y)-(wd*ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_antiderivative1 (y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_antiderivative1 (y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; </pre>
	<pre> command window >> pid_antiderivative1 </pre>

noise-suppressing PID controller can be used:

$$G_c(s) = \frac{u(s)}{y_s(s) - y(s)} = k_c + \frac{k_c}{\tau_i s} + \frac{k_c \tau_d s}{1 + \alpha \tau_d s} \quad (4.88)$$

$$u_p(k) = k_c(y_s(k) - y(k)) \quad \text{proportional part} \quad (4.89)$$

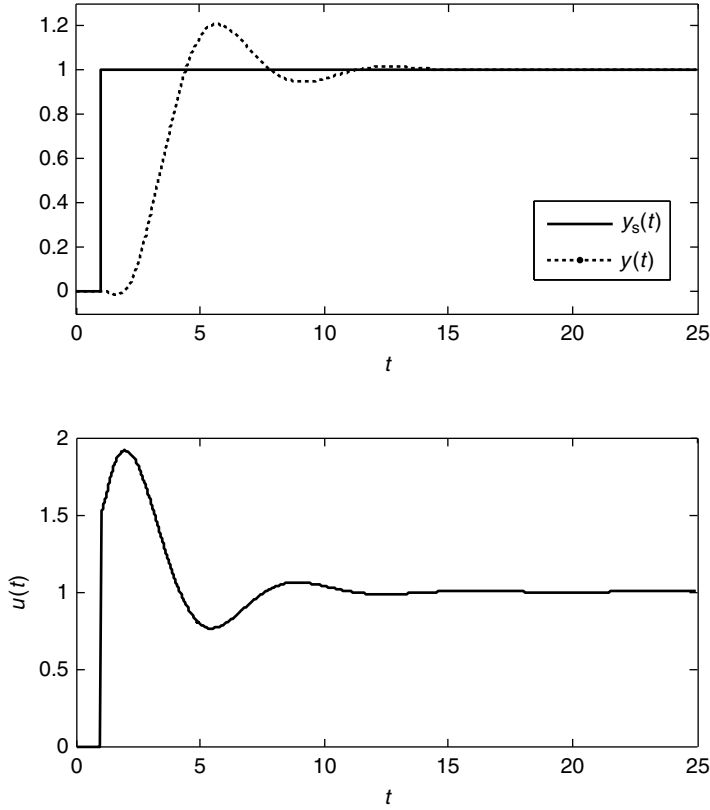


Figure 4.13 Simulation result of Table 4.11.

$$u_i(k) = u_i(k-1) + \frac{k_c}{\tau_i} (y_s(k) - y(k)) \Delta t \quad \text{integral part} \quad (4.90)$$

$$u_d^{\text{nf}}(k) = k_c \tau_d \frac{(y_s(k) - y(k)) - (y_s(k-1) - y(k-1))}{\Delta t} \quad (4.91)$$

$$u_d(k) = u_d(k-1) + \frac{\Delta t}{\alpha \tau_d} (u_d^{\text{nf}}(k-1) - u_d(k-1)) \quad \text{derivative part} \quad (4.92)$$

$$u(k) = u_p(k) + u_i(k) + u_d(k) \quad (4.93)$$

Equation (4.92) can be derived by applying the Euler method to $u_d(s) = u_d^{\text{nf}}(s)/(\alpha \tau_d s + 1)$, and equivalently $du_d(t)/dt + u_d(t)/(\alpha \tau_d) = u_d^{\text{nf}}(t)/(\alpha \tau_d)$ in a straightforward manner. Usually, α ranges from 0.05 to 0.25. Better robustness to noise is obtained for a bigger α value, but the control performance degrades more as the α value increases.

Table 4.12 MATLAB code to simulate a closed-loop control system with a two-degree-of-freedom PID controller.

pid_tdof1.m	g_pid_tdof1.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.01; n=round(t_final/delta_t); % delta_t for Euler delta_ts=0.5; t_previous=-delta_ts; % sampling time for PID C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; kp=0.6; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if (t>=(t_previous+delta_ts -delta_t*0.5)) % sampling if (t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_ts; u=kc*(kp*ys-y)+s+kc*td* ((ys-y)-(ysb-yb))/delta_ts; ysb=ys; yb=y; % one sampling before t_previous=t; end u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_tdof1(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)','y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_tdof1 (y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre> <p>command window</p> <pre> >> pid_tdof1 </pre>

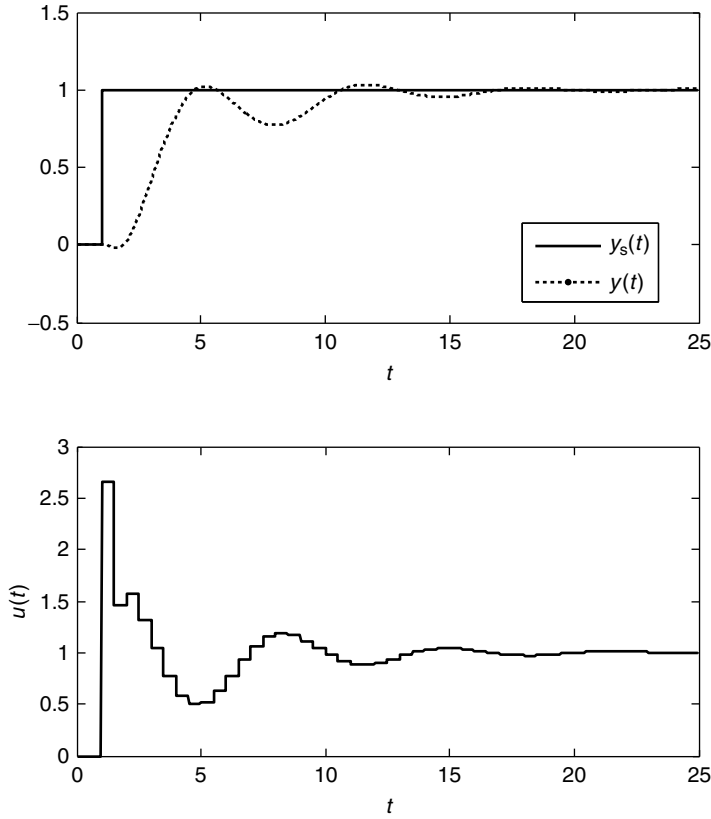


Figure 4.14 Simulation result of Table 4.12.

Example 4.20

Simulate again Figure 4.4 with $\alpha = 0.24$. Compare the closed-loop responses.

Solution The MATLAB code to simulate the noise-suppressing PID controller and the simulation results are given in Table 4.13 and Figure 4.15. Comparing Figure 4.4 with Figure 4.15, the robustness to the noise is improved.

4.4.5 Unified Structure of Proportional–Integral–Derivative Controllers

The two-degree-of-freedom and noise-suppressing PID controllers can be integrated to the following unified form:

$$u(s) = k_c(w_p y_s(s) - y(s)) + \frac{k_c}{\tau_i s} (y_s(s) - y(s)) + \frac{k_c \tau_d s (w_d y_s(s) - y(s))}{1 + \alpha \tau_d s} \quad \text{noninteracting} \quad (4.94)$$

$$u(s) = k_c^i \left(w_p + \frac{1}{s \tau_i^i} \right) \left(\frac{1 + w_d \tau_d^i s}{1 + \alpha \tau_d^i s} \right) y_s(s) - k_c^i \left(1 + \frac{1}{s \tau_i^i} \right) \left(\frac{1 + \tau_d^i s}{1 + \alpha \tau_d^i s} \right) y(s) \quad \text{interacting} \quad (4.95)$$

Table 4.13 MATLAB code to simulate a closed-loop control system with a noise-suppressing PID controller.

pid_nsl.m	g_pid_nsl.m
<pre> clear; t=0.0; t_final=25.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; ud=0.0; delta_t=0.01;n=round(t_final/delta_t); C=[0 0 1]; theta=0.2; % time delay h_u=zeros(1,1000); n_theta=round(theta/delta_t); kc=1.5; ti=3.0; td=0.5; s=0.0; alpha=0.24; rand('seed',0); noise=0.2 *(rand(1,n)-0.5); for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; ud0=kc*td*((ys-y)-(ysb-yb))/delta_t; ud=ud+(ud0-ud)*delta_t/(alpha*td); u=kc*(ys-y)+s+ud; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_pid_nsl(y,x,h_u(1000-n_theta)); x=x+dx_dt*delta_t; y=C*x+noise(i); t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> function [dx_dt]=g_pid_nsl (y,x,u) A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 -0.3 0.0]'; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> pid_nsl </pre>

$$u(s) = k_c^p(w_p y_s(s) - y(s)) + k_i^p \frac{y_s(s) - y(s)}{s} + \frac{k_d^p s(w_d y_s(s) - y(s))}{1 + \alpha \tau_d s} \quad \text{parallel} \quad (4.96)$$

Also, it is straightforward to add the anti-windup techniques to the unified structure by modifying the integral part of $u_i(t)$ to incorporate the conditional integration or the back calculation.

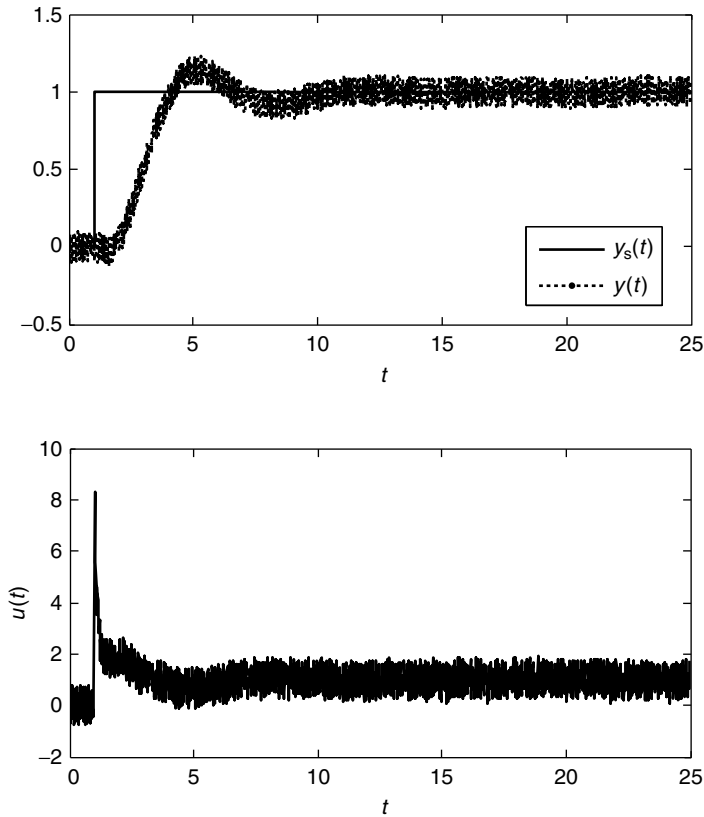


Figure 4.15 Simulation result of Table 4.13.

Problems

- 4.1 Explain the PID controller and the tuning parameters.
- 4.2 The upper limit and the lower limit of the control output are 20 and 4 respectively and the proportional gain of the PID controller is 8. Find the PB of the PID controller.
- 4.3 Draw the response of a PID controller for a unit step error. Explain the effects of the tuning parameters on the response.
- 4.4 Simulate the step setpoint change response for the SOPTD process $G(s) = \exp(-0.3s)/(s + 1)^2$ controlled by a PID controller for which the tuning parameters are $k_c = 1.5$, $\tau_i = 3.0$, $\tau_d = 0.5$.
 - (a) For a sampling time of $\Delta t = 0.01$.
 - (b) For a sampling time of $\Delta t = 0.5$.
 - (c) For a sampling time of $\Delta t = 0.01$ and uniformly distributed random measurement noise between -0.05 and 0.05 .
 - (d) For a sampling time of $\Delta t = 0.2$ and uniformly distributed random measurement noise between -0.05 and 0.05 .

4.5 Explain the roles of the three parts of a PID controller.

4.6 Find the offset for the case that the process $G(s) = \exp(-0.1s)(-0.2s + 1)/(s^2 + 3s + 1)$ controlled by the following controllers for an unit step setpoint change:

- (a) P controller $G_c(s) = 1.5$
- (b) PI controller $G_c(s) = 1.5 + \frac{1.5}{3.0s}$
- (c) PD controller $G_c(s) = 1.5 + 1.5 \times 0.5s$
- (d) PID controller $G_c(s) = 1.5 + \frac{1.5}{3.0s} + 1.5 \times 0.5s$

4.7 Find the offset for the case that the process $G(s) = \exp(-0.2s)/s(s + 1)$ controlled by the following controllers with a unit step setpoint change:

- (a) P controller $G_c(s) = 1.0$
- (b) PI controller $G_c(s) = 1.0 + \frac{1.0}{5.0s}$
- (c) PD controller $G_c(s) = 1.0 + 1.0 \times 0.5s$
- (d) PID controller $G_c(s) = 1.5 + \frac{1.5}{5.0s} + 1.0 \times 0.5s$

4.8 Find the I part ($k_c \int_0^t (y_s(\tau) - y(\tau)) d\tau / \tau_i$) of a PID controller at steady state for the case that the following processes are controlled by a PID controller with a unit step setpoint change:

- (a) $G(s) = \frac{3}{s + 1}$
- (b) $G(s) = \frac{3\exp(-0.5s)}{s + 1}$
- (c) $G(s) = \frac{3(0.1s + 1)(-0.1s + 1)}{(s + 1)^5}$
- (d) $G(s) = \frac{(s + 3)\exp(-0.1s)}{(s + 2)^2}$
- (e) $G(s) = \frac{10}{s(s + 1)^3}$
- (f) $G(s) = \frac{(-s + 1)\exp(-0.5s)}{s(s + 2)^3}$

4.9 Explain the integral windup phenomenon of a PID controller.

4.10 Simulate the closed-loop control system of which the process is $G(s) = 2.0 \exp(-0.3s)/(2.5s^2 + 5.0s + 2)$ and the controller is a PID controller with $k_c = 3.38$, $\tau_i = 2.60$ and $\tau_d = 0.54$. The setpoint changes from 0 to 1 at $t = 1.0$. Also, discuss the effects of the anti-windup techniques.

- (a) No limits to the range of the control output.
- (b) $u_{\max} = 1.2$ and $u_{\min} = -1.2$, no anti-windup technique.
- (c) $u_{\max} = 1.2$ and $u_{\min} = -1.2$, conditional integration technique.
- (d) $u_{\max} = 1.2$ and $u_{\min} = -1.2$, back-calculation technique.

- 4.11 Find the noninteracting PID controller and the parallel PID controller equivalent to the interacting PID controller $G_c(s) = 1.5[1 + (1/10s)](1 + 2.5s)$.
- 4.12 Find the interacting PID controller and the parallel PID controller equivalent to the noninteracting PID controller $G_c(s) = 1.5 + (1/10s) + 2.5s$.
- 4.13 Simulate the closed-loop control system of which the process is $G(s) = 2.0 \exp(-0.3s)/(2.5s^2 + 5.0s + 2.0)$ and the controller is a noninteracting PID controller with $k_c = 3.38$, $\tau_i = 2.60$ and $\tau_d = 0.54$. The setpoint changes from 0 to 1 at $t = 1.0$. Also, discuss the effects of the tuning parameters.
- Two-degree-of-freedom PID controller with $w_p = 1.0$ and $w_d = 1.0$.
 - Two-degree-of-freedom PID controller with $w_p = 1.0$ and $w_d = 0.0$.
 - Two-degree-of-freedom PID controller with $w_p = 0.7$ and $w_d = 0.0$.
- 4.14 Simulate Problem 4.13 with measurement noise between -0.05 and 0.05 and noise-suppressing PID controller.
- 4.15 Apply the following controllers to the virtual process of Process 5 (refer to the Appendix for details) with the scan time of 0.2 and obtain the unit step setpoint change response:
- Noninteracting PID controller with $k_c = 1.81$, $\tau_i = 16.3$, $\tau_d = 5.90$, $\Delta t = 0.8$, $u_{\max} = \infty$, $u_{\min} = -\infty$.
 - Noninteracting PID controller with $k_c = 1.81$, $\tau_i = 16.3$, $\tau_d = 5.90$, $\Delta t = 0.8$, $u_{\max} = 2.2$, $u_{\min} = 0.0$.
 - Noninteracting PID controller with $k_c = 1.81$, $\tau_i = 16.3$, $\tau_d = 5.90$, $\Delta t = 0.8$, conditional integration, $u_{\max} = 2.2$, $u_{\min} = 0.0$.
 - Noninteracting PID controller with $k_c = 1.81$, $\tau_i = 16.3$, $\tau_d = 5.90$, $\Delta t = 0.2$, $u_{\max} = \infty$, $u_{\min} = -\infty$.
 - Noninteracting PID controller with $k_c = 1.81$, $\tau_i = 16.3$, $\tau_d = 5.90$, $\Delta t = 0.2$, noise-suppressing PID controller, $u_{\max} = \infty$, $u_{\min} = -\infty$.

Bibliography

- Astrom, K.J. and Hagglund, T. (1995) *PID controllers*, Instrument Society of America, NC.
- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
- Stephanopoulos, G. (1984) *Chemical Process Control - An Introduction to Theory and Practice*, Prentice-Hall.

5

Proportional–Integral–Derivative Controller Tuning

The tuning parameters of the PID controller should be set with in-depth consideration of the process dynamics. Otherwise, acceptable control performances cannot be achieved. Poor tuning parameters would result in very sluggish or unstable responses. In this chapter, simple process identification methods are introduced to obtain the process model in the form of the frequency response or low order plus time delay. Also, various tuning methods are discussed to demonstrate how to estimate the tuning parameters of the PID controller on the basis of the process model.

5.1 Trial-and-Error Tuning

Trial-and-error tuning is used to determine the tuning parameters of a PID controller by inspecting the dynamic behavior of the controlled process output. It is very important to understand the effects of the tuning parameters on the behavior of the process output for successful trial-and-error tuning. The PID controller usually shows the following dynamic behaviors with respect to the tuning parameters for the step setpoint change.

- Behavior 1. For a step setpoint change, if the process output shows a big oscillation, as shown in Figure 5.1, then the proportional gain k_c is too large. Here, the oscillation is centered on the setpoint.
- Behavior 2. For a step setpoint change, if the controlled process output shows an overdamped response, as shown in Figure 5.2, then the proportional gain k_c of the PID controller is too small.
- Behavior 3. For a positive step setpoint change, if the process output oscillates and the process output stays above the setpoint longer than under the setpoint, as shown in Figure 5.3, then the integral time τ_i is too small (that is, the integral action is too strong).
- Behavior 4. For a positive step setpoint change, if the process output oscillates and the process output stays under the setpoint longer than above the setpoint, as shown in Figure 5.4,

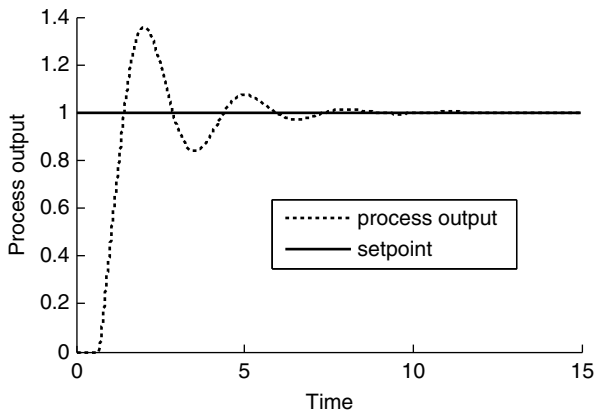


Figure 5.1 Typical closed-loop responses for too large a proportional gain.

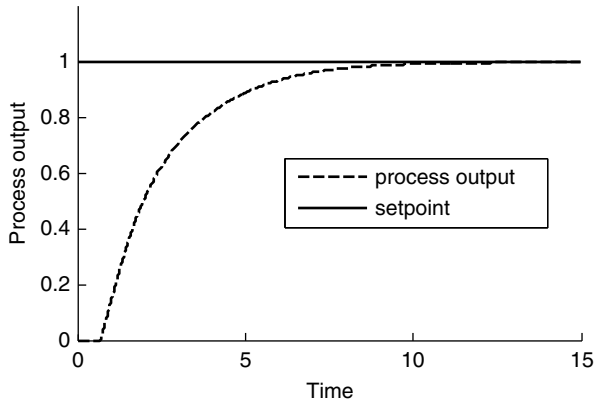


Figure 5.2 Typical closed-loop response for too small a proportional gain.

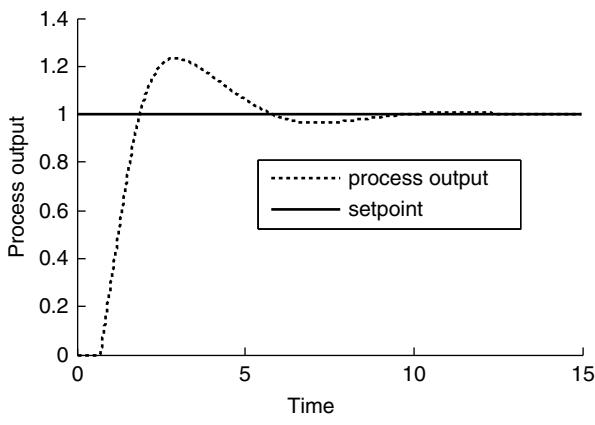


Figure 5.3 Typical closed-loop responses for too small an integral time with a proper proportional gain and derivative time.

- then the integral time τ_i is too large (that is, the integral action is too weak). For open-loop stable processes, $y_s/k = (k_c/\tau_i) \int_0^\infty (y_s(t) - y(t)) dt$ is valid if the offset is zero, where k , y_s and y denote the static gain, the step setpoint and the process output respectively. This means that the amount of the integral part of the PID controller is fixed on y_s/k at steady state. So, if the integral amount from the starting point to the rise time is bigger than y_s/k due to a small τ_i value, then the integral term will be reduced to y_s/k by allowing the process output to stay above the setpoint longer than under the setpoint, as shown in Figure 5.3. If the integral amount from the starting point to the first peak is much smaller than y_s/k due to a large τ_i value, then the integral term will be accumulated to y_s/k by allowing the process output to stay under the setpoint longer than above the setpoint, as shown in Figure 5.4.
- Behavior 5. For a step setpoint change, if the process output shows a high-frequency oscillation (many peaks) from the start to the steady state, as shown in Figure 5.5, then the

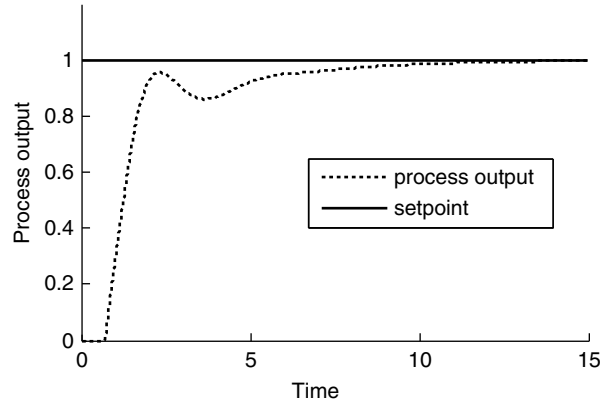


Figure 5.4 Typical closed-loop response for too large an integral time with a proper proportional gain and derivative time.

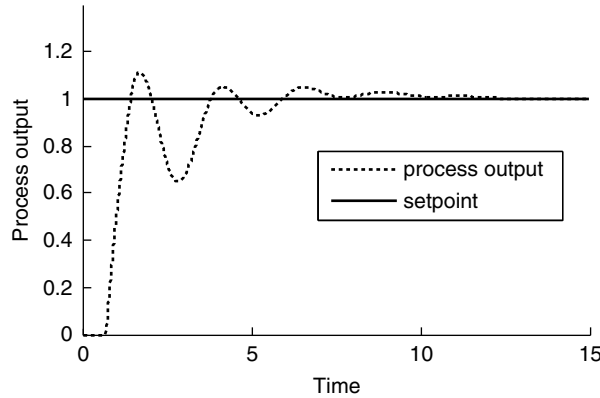


Figure 5.5 Typical closed-loop response for too large a derivative time.

derivative time τ_d is too large. This phenomenon is due to the amplification of a high-frequency signal by the strong derivative part.

Now, the operator can tune the PID controller in the trial-and-error manner by adjusting the tuning parameters in the direction of removing the above-mentioned five dynamic behaviors. The final important thing to be kept in mind for successful trial-and-error tuning is to sustain the proportional gain as large as possible. The closed-loop dynamics tend to become slower and slower during trial-and-error tuning if the focus is on removing the five dynamic behaviors without trying to sustain a large proportional gain.

5.2 Simple Process Identification Methods

Because the parameters of the PID controller should be tuned on the basis of the process dynamics, the process model should first be identified for the tuning. So, simple process identification methods, such as the continuous-cycling method, the PRC method and the frequency test are introduced in this section before moving to various tuning methods.

5.2.1 Continuous-Cycling Method

The continuous-cycling method is used to estimate the ultimate frequency and the ultimate gain of the process. Consider the following control system with a proportional (P) controller.

In the continuous-cycling method, first, the P controller should be implemented as shown in Figure 5.6. Second, the control engineer increases the controller gain and simultaneously changes the setpoint. If the process output diverges, then the engineer reduces the controller gain and vice versa. This procedure repeats until the process output oscillates continuously, as shown in Figure 5.7. Then, the proportional gain of the P controller after 60 s is the ultimate gain of the process. The ultimate period of the process is the period of the continuous-cycling.

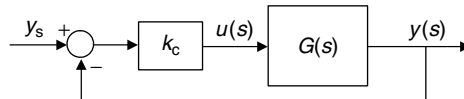


Figure 5.6 Control system with a proportional controller.

The procedure of this method is simple and has been accepted by control engineers. However, it is a trial-and-error approach, resulting in a long identification time. Also, it is undesirable to push the control system to the marginally stable zone from the safety point of view.

In real industrial plants, the engineer can usually set the upper and lower limits on the control output, which plays an important role in preventing divergences. The continuous-cycling test can be performed in a safer way by appropriately setting the upper and lower limits to the control output, as shown in Figure 5.8.

As shown in Figure 5.8, the magnitude of the oscillation of the process output is limited by setting the upper/lower limit on the control output. If the proportional gain is smaller than the ultimate gain, then the control output converges. If the proportional gain is bigger than the

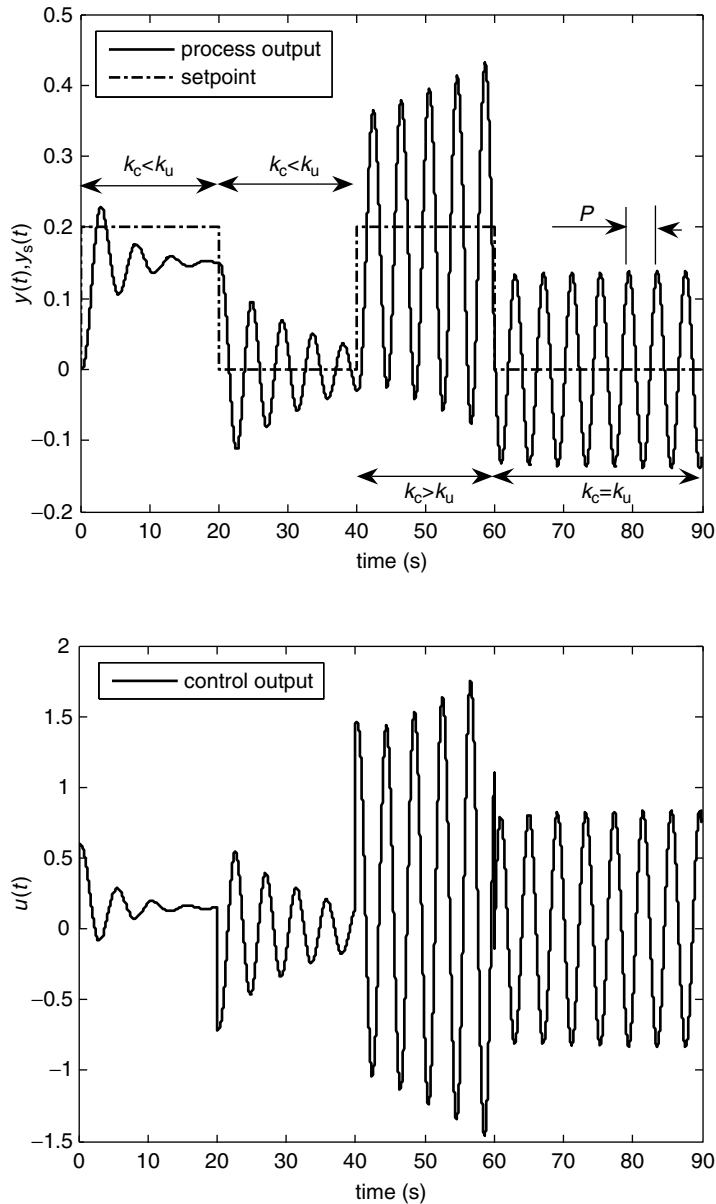


Figure 5.7 Identification using the continuous-cycling method.

ultimate gain, then at least one side of the oscillation of the control output is saturated to the upper or lower limit. Meanwhile, one side of the oscillation contacts slightly with one of the limits (but not saturated) if the proportional gain is the same as the ultimate gain.

Let us justify why the proportional gain and the period of the continuous-cycling in Figure 5.7 are the ultimate gain and the ultimate period respectively. In Figure 5.7, the process

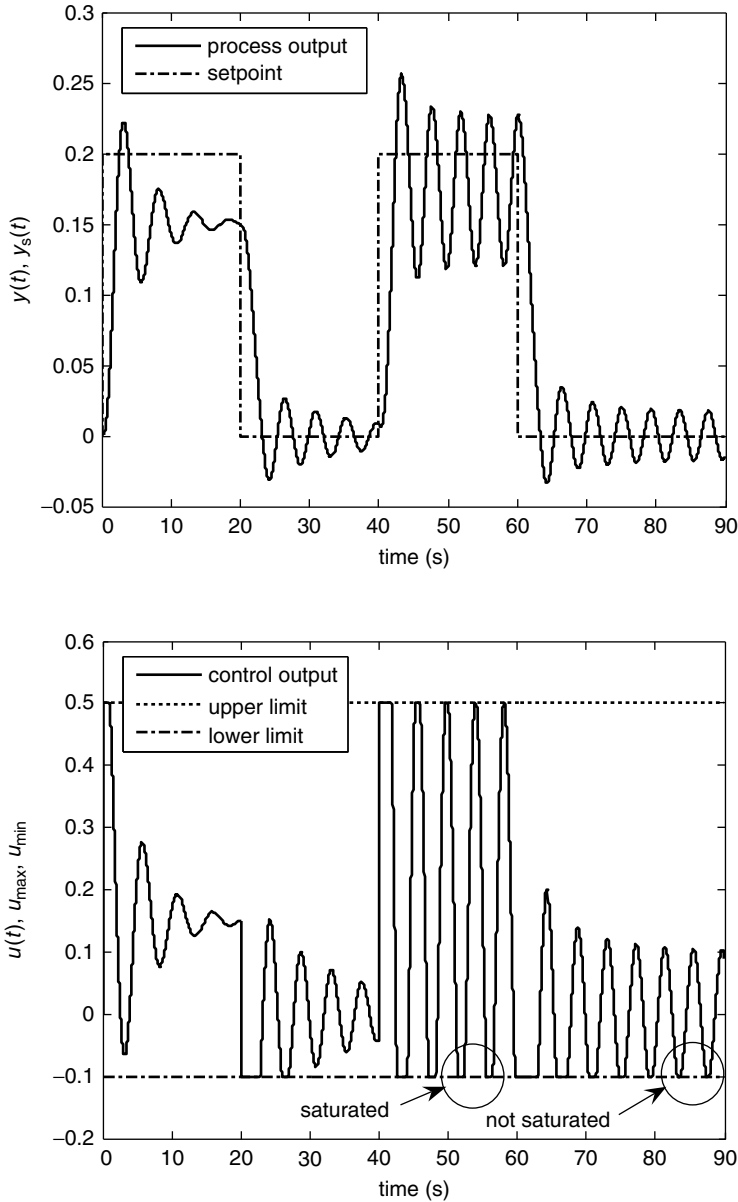


Figure 5.8 Continuous-cycling test with upper and lower limits on the control output.

output is a sine signal like $y(t) = -c \sin(\omega t) + d$ since the input/output signals of linear systems in continuous cycling are always sine signals. Then, the control output is $u(t) = k_c c \sin(\omega t) - k_c d + k_c y_s$ from $u(t) = k_c(y_s - y(t))$. The signs of the two sine signals $u(t)$ and $y(t)$ are opposite. So, the period of continuous cycling is the ultimate period (equivalently, $\angle G(i\omega) = -\pi$). Also, the ultimate amplitude ratio of $|G(i\omega)|$ is $c/k_c c = 1/k_c$ and the ultimate gain is k_c (equivalently, $k_c = 1/|G(i\omega)|$).

5.2.2 Process Reaction Curve Method and Frequency Test

The PRC method and the frequency test method were introduced in Chapter 3. They provide the FOPTD model and the frequency response data (ultimate frequency and ultimate gain). For details, refer to Chapter 3.

5.2.3 Advanced Process Identification Methods

Advanced process identification methods are summarized in Part Three. They can provide complex high-order plus time-delay models and complex frequency response models. For details, refer to Part Three.

If the process model is obtained with the process identification methods, then the parameters of the PID controller are ready to be tuned.

5.3 Ziegler–Nichols Tuning Rule

The Ziegler–Nichols (ZN) tuning rule (Ziegler and Nichols, 1942) uses the ultimate gain and the ultimate period of the process. Table 5.1 provides the tuning parameters of the PID controller for the given ultimate data set of the process.

Table 5.1 ZN tuning rule.

Controller	Tuning parameters ^a		
	k_c	τ_i	τ_d
P	$k_u/2.0$	—	—
PI	$k_u/2.2$	$p_u/1.2$	—
PID	$k_u/1.7$	$p_u/2.0$	$p_u/8.0$

^a k_u and p_u denote the ultimate gain and the ultimate period of the process respectively.

The tuning rule is very simple and needs only the ultimate information, which can be estimated easily by simple identification methods, such as the continuous-cycling method and relay feedback identification method (which will be introduced in Part Three). The ZN tuning rule shows acceptable control performances for the usual processes. However, because the ZN tuning rule uses only the ultimate data of the process, it shows poor control performances for underdamped or large time-delay processes, because the process has unusual frequency response characteristics in the low-frequency region.

Example 5.1

Obtain the tuning parameters of a PID controller if the ultimate gain and the ultimate frequency are $k_u = 0.2$ and $\omega_u = 0.9$ respectively.

Solution $\omega_u = 0.9$ means that the ultimate period is $p_u = 2\pi/0.9$. So, the tuning parameters of $k_c = 0.2/1.7$, $\tau_i = \pi/0.9$ and $\tau_d = \pi/(0.9 \times 4)$ are obtained from Table 5.1.

Example 5.2

Obtain the tuning parameters of a PI and a PID controller for the process of which the transfer function is $G(s) = 1/(s + 1)^3$.

Solution The ultimate frequency and the ultimate gain can be estimated if the transfer function of the process is given. Because $G(i1.732) = 1/(i1.732 + 1)^3 = -0.125 + i0.000$, the ultimate frequency is $\omega_u = 1.732$ (equivalently, $p_u = 2\pi/1.732$) and the ultimate gain is $k_u = 1/0.125$. For the detailed descriptions, refer to Chapter 3. So, the tuning parameters of $k_c = 1/(0.125 \times 2.2)$ and $\tau_i = 2\pi/(1.732 \times 1.2)$ are obtained from Table 5.1 for the PI controller. In a similar way, the tuning parameters of the PID controller can be obtained. Table 5.2 and Figure 5.9 show the MATLAB code for the simulations and the simulation results respectively. In Figure 5.9a, the setpoint is changed at $t = 1$ and a step input disturbance of magnitude equal to 1 is entered at $t = 1$ in Figure 5.9b. Here, the step input disturbance is the step-type disturbance added to the process input.

Table 5.2 MATLAB code to simulate a PID control system tuned by the ZN tuning rule.

<pre>zn_model_ex2.m clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_zn_model_ex2(w); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1<wu<w2</pre>	<pre>g_zn_model_ex2.m function [G]=g_zn_model_ex2(w) s=i*w; G=1/(s+1)^3; end</pre>
<pre>while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_zn_model_ex2(w1); g=g_zn_model_ex2(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found pu=2*pi/wu; ku=1/abs(g_zn_model_ex2(wu)); p_kcd=ku/2.0; pi_kcd=ku/2.2; pi_tid=pu/1.2; pid_kcd=ku/1.7; pid_tid=pu/2.0; pid_tdd=pu/8.0; fprintf('P: kcd=%6.3f \n', p_kcd); fprintf('PI: kcd=%6.3f, tid=%6.3f \n', pi_kcd, pi_tid); fprintf('PID: kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', pid_kcd, pid_tid, pid_tdd);</pre>	<pre>g_pid_zn_ex2.m function [next_x y]=g_pid_zn_ex2 (x,delt,u) subdelt=delt/5; n=round(delt/subdelt); A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1; 0; 0]; C=[0 0 1]; delay=0.0; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end</pre>

Table 5.2 (Continued)

	<pre> command window >> zn_model_ex2 P: kcd= 4.000 PI: kcd= 3.636, tid= 3.023 PID: kcdd= 4.706, tid= 1.814, tdd= 0.453 >> pid_zn_ex2 </pre>
<pre> pid_zn_ex2.m kc=pid_kcd; ti=pid_tid; td=pid_tdd; t=0.0; t_final=15.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0; delta_t=0.02; n=round(t_final/delta_t); h_u=zeros(1,500); s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end % setpoint change simulation % if(t>1) dis=1.0; else dis=0.0; end % disturbance rection simulation s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:499 h_u(j)=h_u(j+1); end h_u(500)=u+dis; [x y]=g_pid_zn_ex2(x,delta_t,h_u); t=t+delta_t; end figure(1); plot(t_array,ys_array,t_array,y_array); legend('y_{s} (t)','y(t)'); figure(2); plot(t_array,u_array); </pre>	

5.4 Internal Model Control Tuning Rule

The objective of the internal model control (IMC) tuning rule (Morari and Zafriou, 1989) is to match the control performance of the PID controller with that of the IMC controller. It needs the following FOPTD model:

$$G_m(s) = \frac{k \exp(-\theta s)}{\tau s + 1} \quad (5.1)$$

where k , τ and θ denote the static gain, the time constant and the time delay respectively. The FOPTD model can approximate the usual overdamped processes. The model can be obtained by various process identification methods. Refer to Section 5.2 for the detailed descriptions. The IMC tuning rule determines the tuning parameters using the formulas in Table 5.3, where $\lambda \geq 0.25\theta$ for the PID controller and $\lambda \geq 1.7\theta$ for the PI controller. If a smaller value of λ is

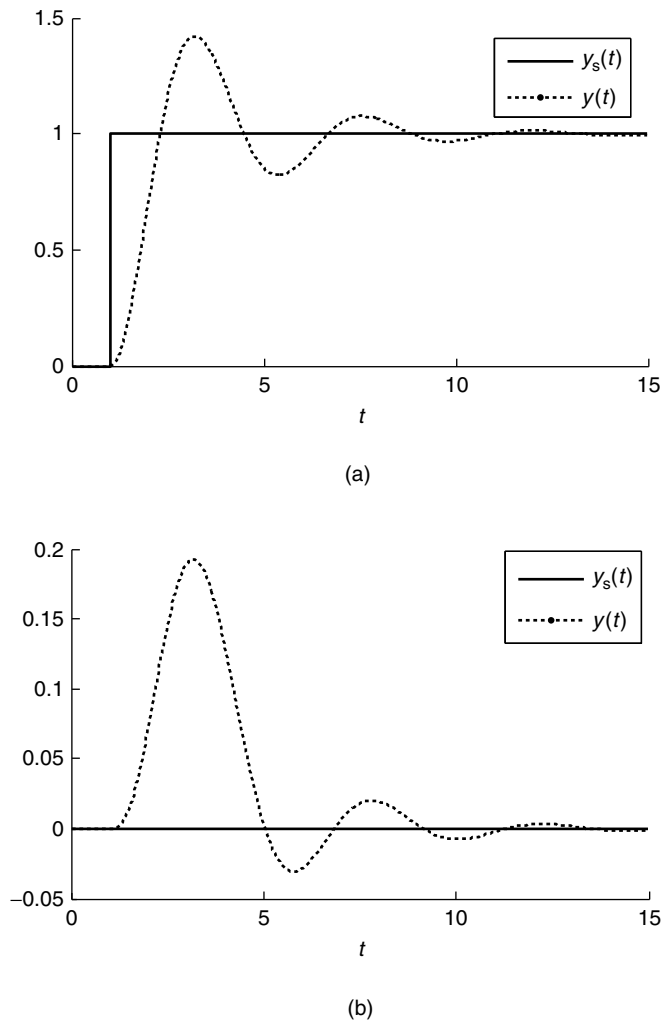


Figure 5.9 Control performances of a PID controller designed by the ZN tuning rule: (a) setpoint change; (b) disturbance rejection.

Table 5.3 IMC tuning rule.

Controller	Tuning parameters		
	kk_c	τ_i	τ_d
PI	$(2\tau + \theta)/2\lambda$	$\tau + \theta/2$	—
PID	$(2\tau + \theta)/2(\lambda + \theta)$	$\tau + \theta/2$	$\tau\theta/(2\tau + \theta)$

chosen, then a faster closed-loop response is obtained. However, too small a λ value results in an oscillatory or unstable closed-loop response. If the model (5.1) is accurate, then the tuning parameters with $\lambda = 0.25\theta$ show good control performances and robustness for the step setpoint change.

The IMC tuning rule shows excellent control performances for a step setpoint change. Meanwhile, it shows sluggish control performances for step input disturbance rejection. Here, the step input disturbance is the step-type disturbance added to the process input.

The FOPTD model has a structural limitation in representing underdamped or high-order processes. Thus, the IMC tuning rule based on the FOPTD model shows poor control performances for unusual processes, such as underdamped or high-order processes.

Example 5.3

Obtain the tuning parameters of a PID controller for the process of which the transfer function is $G(s) = 1.5\exp(-0.3s)/(1.2s + 1)$ using the IMC tuning rule. And simulate the closed-loop response for the step setpoint change and the step input disturbance rejection.

Solution $\lambda = 0.25 \times 0.3$ is chosen and then the tuning parameters $k_c = (2 \times 1.2 + 0.3)/[2(0.25 \times 0.3 + 0.3) \times 1.5]$, $\tau_i = 1.2 + 0.3/2$ and $\tau_d = (1.2 \times 0.3)/(2 \times 1.2 + 0.3)$ are obtained from Table 5.3. Table 5.4 and Figure 5.10 are the MATLAB code for the simulation and the simulation results. The step setpoint is changed at $t = 1$ in Figure 5.10a and the step input disturbance of magnitude equal to 1 enters at $t = 1$ in Figure 5.10b. It is clear that the control action for the step input disturbance rejection is more sluggish than that for the step setpoint change.

5.5 Integral of the Time-Weighted Absolute Value of the Error Tuning Rule for a First-Order Plus Time-Delay Model (ITAE-1)

The integral of the time-weighted absolute value of the error tuning rule for an FOPTD model (ITAE-1) tuning rule provides the tuning parameters minimizing the following integral of the time-weighted absolute value of the error (ITAE).

$$\text{ITAE} = \int_0^{\infty} t |y_s(t) - y(t)| dt \quad (5.2)$$

Lopez *et al.* (1967) provides the equations in Table 5.5 to approximate the optimal tuning parameters minimizing (5.2).

The parameters tuned by the ITAE-1-disturbance method for the step input disturbance rejection are almost the same as the optimal tuning parameters. But the ITAE-1 setpoint shows a sluggish control action compared with that of the optimal tuning parameters for the step setpoint change.

Example 5.4

Obtain the tuning parameters of a PID controller for the process of which the transfer function is $G(s) = 1.5 \exp(-0.3s)/(1.2s + 1)$ using the ITAE-1 tuning rule. Also, simulate the closed-loop response for the step setpoint change and the step input disturbance rejection.

Table 5.4 MATLAB code to simulate a PID control system tuned by the IMC tuning rule.

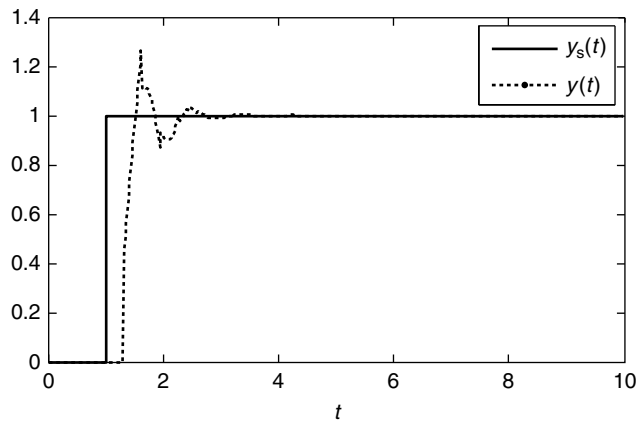
<pre> imc_tune_ex1.m clear; k=1.5; t=1.2; th=0.3; r=1.7*th; pi_kcs=(2*t+th)/2/r/k; pi_tis=t+th/2; r=0.25*th; pid_kcs=(2*t+th)/(2/(r+th)/k); pid_tis=t+th/2; pid_tds=t*th/(2*t+th); fprintf('PI: kcs=%6.3f, tis=%6.3f\n', pi_kcs, pi_tis); fprintf('PID: kcds=%6.3f, tis=%6.3f, tds=%6.3f\n', pid_kcs, pid_tis, pid_tds); </pre>	<pre> g_pid_imc_ex1.m function [next_x y]=g_pid_imc_ex1 (x,delt,u) subdelt=delt/5; n=round(delt/subdelt); A=[-1.0/1.2]; B=[1.5/1.2]; C=[1]; delay=0.3; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end </pre> <hr/> <pre> command window >> imc_tune_ex1 PI: kcs= 1.765, tis= 1.350 PID: kcds= 2.400, tis= 1.350, tds= 0.133 >> pid_imc_ex1 </pre>
--	---

```

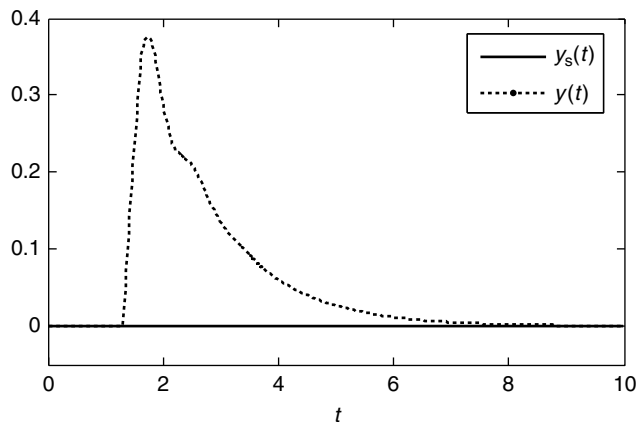
pid_imc_ex1.m
kc=pid_kcs; ti=pid_tis; td=pid_tds;
t=0.0; t_final=10.0; x=[0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0;
delta_t=0.02; n=round(t_final/delta_t); h_u=zeros(1,500); s=0.0;
for i=1:n
    t_array(i)=t; y_array(i)=y; ys_array(i)=ys;
    if (t>1) ys=1.0; else ys=0.0; end % setpoint change simulation
% if (t>1) dis=1.0; else dis=0.0; end % disturbance rection
    simulation s=s+(kc/ti)*(ys-y)*delta_t;
    u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t;
    ysb=ys; yb=y; % one sampling before
    u_array(i)=u;
    for j=1:499 h_u(j)=h_u(j+1); end
    h_u(500)=u+dis; [x y]=g_pid_imc_ex1(x,delta_t,h_u);
    t=t+delta_t;
end
figure(3); plot(t_array,ys_array,t_array,y_array);
legend('y_{s}(t)', 'y(t)');
figure(2); plot(t_array,u_array);

```

Solution $k_c = 2.090$, $\tau_i = 1.580$ and $\tau_d = 0.102$ for the step setpoint change and $k_c = 3.362$, $\tau_i = 0.512$ and $\tau_d = 0.115$ for the step input disturbance are obtained from Table 5.5. Table 5.6 and Figure 5.11 show the MATLAB code and the simulation results respectively. The step setpoint is changed at $t = 1$ in Figure 5.11a and the step input disturbance of magnitude equal to 1 enters at $t = 1$ in Figure 5.11b. The step input disturbance is a step-type disturbance added to



(a)



(b)

Figure 5.10 Control performances of a PID controller designed by the IMC tuning rule: (a) setpoint change; (b) disturbance rejection.

Table 5.5 ITAE-1 tuning rule for an FOPTD model.

Controller	Tuning parameters		
	kk_c	τ/τ_i	τ_d/τ
PI-setpoint	$0.586(\theta/\tau)^{-0.916}$	$1.030-0.165(\theta/\tau)$	—
PID-setpoint	$0.965(\theta/\tau)^{-0.850}$	$0.796-0.1465(\theta/\tau)$	$0.308(\theta/\tau)^{0.929}$
PI-disturbance	$0.859(\theta/\tau)^{-0.977}$	$0.674(\theta/\tau)^{-0.680}$	—
PID-disturbance	$1.357(\theta/\tau)^{-0.947}$	$0.842(\theta/\tau)^{-0.738}$	$0.381(\theta/\tau)^{0.995}$

Table 5.6 MATLAB code to simulate a PID control system tuned by the ITAE-1 tuning rule.

<pre> itael_tune_ex1.m clear; k=1.5; t=1.2; th=0.3; pi_kcs=0.586*(th/t)^(-0.916)/k; pi_tis=t/(1.030-0.165*th/t); pid_kcs=0.965*(th/t)^(-0.850)/k; pid_tis=t/(0.796-0.1465*th/t); pid_tds=t*0.308*(th/t)^0.929; pi_kcd=0.859*(th/t)^(-0.977)/k; pi_tid=t/(0.674*(th/t)^(-0.680)); pid_kcd=1.357*(th/t)^(-0.947)/k; pid_tid=t/(0.842*(th/t)^(-0.738)); pid_tdd=t*0.381*(th/t)^0.995; fprintf('PI-setpoint: kcs=%6.3f, tis=%6.3f \n',pi_kcs,pi_tis); fprintf('PID-setpoint: kcds=%6.3f, tis=%6.3f, tds=%6.3f \n',pid_kcs, pid_tis,pid_tds); fprintf('PI-disturbance: kcd=%6.3f, tid=%6.3f \n',pi_kcd,pi_tid); fprintf('PID-disturbance: kcdd=%6.3f, tid=%6.3f, tdd=%6.3f \n', pid_kcd,pid_tid,pid_tdd); </pre>	
<pre> pid_itaex1.m kc=pid_kcs; ti=pid_tis; td=pid_tds; % setpoint change simulation %kc=pid_kcd; ti=pid_tid; td=pid_tdd; % disturbance rection simulation t=0.0; t_final=10.0; x=[0]'; y=0.0; yb=0.0; ysb=0.0; dis=0.0; delta_t=0.02; n=round(t_final/delta_t); h_u=zeros(1,500); s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end % setpoint change simulation % if(t>1) dis=1.0; else dis=0.0; end % disturbance rection simulation s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:499 h_u(j)=h_u(j+1); end h_u(500)=u+dis; [x y]=g_pid_itaex1(x,delta_t,h_u); t=t+delta_t; end figure(1); hold on; plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)','y(t)'); figure(2); plot(t_array,u_array); </pre>	
<pre> g_pid_itaex1.m function [next_x y] = g_pid_itaex1(x,delt,u) subdelt=delt/5; n=round(delt/subdelt); A=[-1.0/1.2]; B=[1.5/1.2]; C=[1]; delay=0.3; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end </pre>	<pre> command window >> itael_tune_ex1 PI-setpoint: kcs= 1.391, tis= 1.214 PID-setpoint: kcds= 2.090, tis= 1.580, tds= 0.102 PI-disturbance: kcd= 2.219, tid= 0.694 PID-disturbance: kcdd= 3.362, tid= 0.512, tdd= 0.115 </pre>

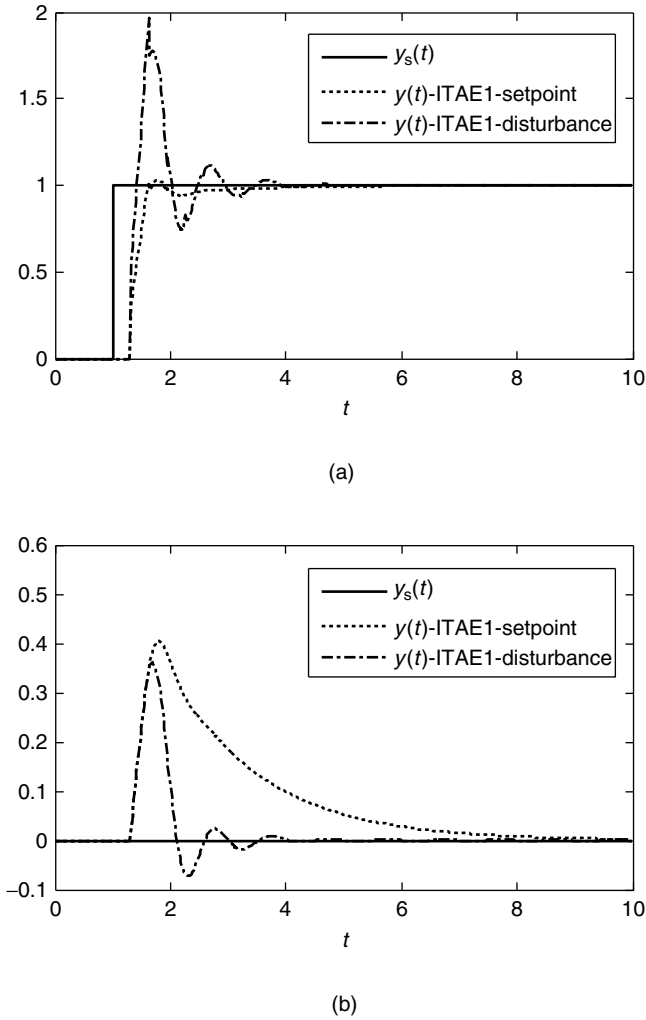
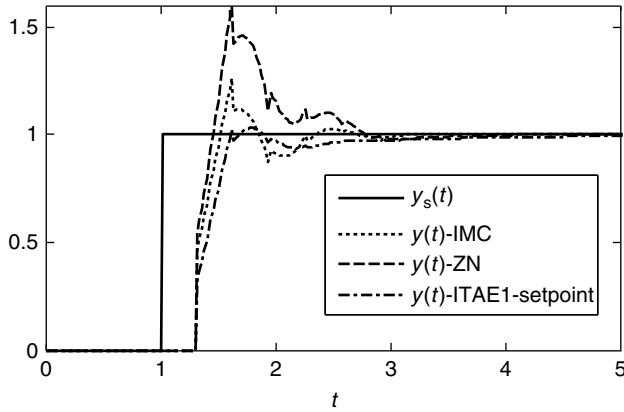


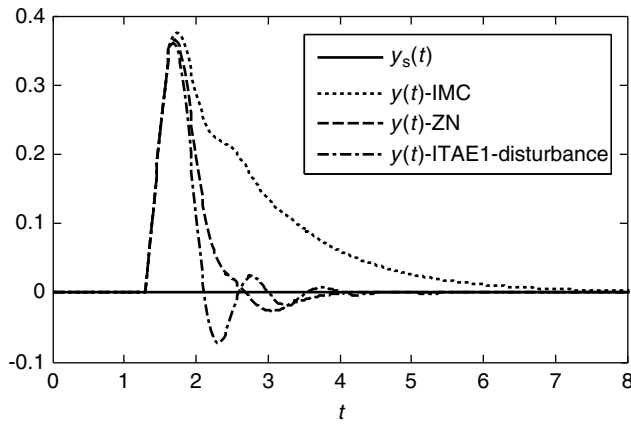
Figure 5.11 Control performances of a PID controller designed by the ITAE-1 tuning rule: (a) setpoint change; (b) disturbance rejection.

the process input. The dynamic behaviors of the step input disturbance and the step output disturbance are totally different. Usually, the PID controller tuned for the step input disturbance shows more aggressive control action compared with the PID controller tuned for the step setpoint change. So, the ITAE-1-setpoint shows a too sluggish control action for the step input disturbance and the ITAE-1-disturbance shows a too aggressive control action for the step setpoint change.

Figure 5.12 compares the control performances of the tuning rules for the FOPTD model. Here, the process is $G(s) = 1.5 \exp(-0.3s)/(1.2s + 1)$. As shown in Figure 5.12, the IMC tuning rule and the ITAE-1-disturbance show superior control performances for the setpoint change and the step input disturbance rejection problems respectively. Contrarily, the IMC tuning rule shows a sluggish response for the step input disturbance. Also, the ZN tuning rule



(a)



(b)

Figure 5.12 Control performances of PID controllers tuned by ZN, ITAE-1 and IMC tuning rules: (a) step setpoint change; (b) step input disturbance rejection.

shows a poor control performance for the step setpoint change. The IMC tuning rule is recommended to tune the PID controller for the step setpoint change problem and the ITAE-1-disturbance is recommended for the step input disturbance rejection problem.

5.6 Integral of the Time-Weighted Absolute Value of the Error Tuning Rule for a Second-Order Plus Time-Delay Model (ITAE-2)

The SOPTD model of (5.3) can describe the dynamics of high-order or underdamped processes with fairly good accuracy. So, it is clear that the tuning rules on the basis of the SOPTD model are potentially much better than those on the basis of the FOPTD model.

$$G_m(s) = \frac{k \exp(-\theta s)}{\tau^2 s^2 + 2\tau\zeta s + 1} \quad (5.3)$$

Sung *et al.* (1996) proposed the ITAE-2 tuning rule for an SOPTD model. They obtained the optimal tuning parameters by solving the optimization problems of (5.2) for the extensive cases of the SOPTD model. And they fitted the optimal data sets obtained as shown in Table 5.7. ITAE-2 is applicable to all the cases of $0.3 \leq \zeta \leq 5.0$ and $0.05 \leq \theta/\tau \leq 2.0$. ITAE-2 provides almost the same control performances as those of the optimal tuning for both the step setpoint change and the step input disturbance rejection problems.

Table 5.7 ITAE-2 tuning rule for an SOPTD model.

ITAE-2-setpoint	$kk_c = -0.04 + \left[0.333 + 0.949 \left(\frac{\theta}{\tau} \right)^{-0.983} \right] \zeta, \quad \zeta \leq 0.9$ $kk_c = -0.544 + 0.308 \left(\frac{\theta}{\tau} \right) + 1.408 \left(\frac{\theta}{\tau} \right)^{-0.832} \zeta, \quad \zeta > 0.9$
	$\frac{\tau_i}{\tau} = \left[2.055 + 0.072 \left(\frac{\theta}{\tau} \right) \right] \zeta, \quad \frac{\theta}{\tau} \leq 1.0$
	$\frac{\tau_i}{\tau} = \left\{ 1.768 + 0.329 \left(\frac{\theta}{\tau} \right) \right\} \zeta, \quad \frac{\theta}{\tau} > 1.0$
	$\frac{\tau}{\tau_d} = \left\{ 1.0 - \exp \left[- \frac{(\theta/\tau)^{1.060} \zeta}{0.870} \right] \right\} \left[0.55 + 1.683 \left(\frac{\theta}{\tau} \right)^{-1.090} \right]$
ITAE-2-disturbance	$kk_c = -0.670 + 0.297 \left(\frac{\theta}{\tau} \right)^{-2.001} + 2.189 \left(\frac{\theta}{\tau} \right)^{-0.766} \zeta, \quad \frac{\theta}{\tau} < 0.9$
	$kk_c = -0.365 + 0.260 \left(\frac{\theta}{\tau} - 1.400 \right)^2 + 2.189 \left(\frac{\theta}{\tau} \right)^{-0.766} \zeta, \quad \frac{\theta}{\tau} \geq 0.9$
	$\frac{\tau_i}{\tau} = 2.212 \left(\frac{\theta}{\tau} \right)^{0.520} - 0.300, \quad \frac{\theta}{\tau} < 0.4$
	$\frac{\tau_i}{\tau} = -0.975 + 0.910 \left(\frac{\theta}{\tau} - 1.845 \right)^2 + \left\{ 1 - \exp \left[- \frac{\zeta}{0.150 + 0.330(\theta/\tau)} \right] \right\} \times \left[5.250 - 0.880 \left(\frac{\theta}{\tau} - 2.800 \right)^2 \right], \quad \frac{\theta}{\tau} \geq 0.4$
	$\frac{\tau}{\tau_d} = -1.900 + 1.576 \left(\frac{\theta}{\tau} \right)^{-0.530} + \left\{ 1 - \exp \left[- \frac{\zeta}{-0.15 + 0.939(\theta/\tau)^{-1.121}} \right] \right\} \times \left[1.45 + 0.969 \left(\frac{\theta}{\tau} \right)^{-1.171} \right]$

Example 5.5

Obtain the tuning parameters of a PID controller for the process of which the transfer function is $G(s) = 1.5 \exp(-0.3s)/(2.5s^2 + 5.0s + 2)$ using the ITAE-2 tuning rule.

Solution Because $G(s) = 1.5 \exp(-0.3s)/(2.5s^2 + 5.0s + 2)$ is equivalent to $G(s) = 0.75 \exp(-0.3s)/(1.118^2s^2 + 2 \times 1.118 \times 1.118s + 1)$, $k_c = 5.656$, $\tau_i = 2.593$ and $\tau_d = 0.538$ for the step setpoint change and $k_c = 13.55$, $\tau_i = 0.912$ and $\tau_d = 0.409$ for the step input disturbance change are obtained from Table 5.8. Figure 5.13 shows the control performances of the ITAE-2-setpoint and ITAE-2-disturbance.

Table 5.8 MATLAB code to simulate a PID control system tuned by the ITAE-2 tuning rule.

```

                                itae2_tune_ex1.m
k=0.75; t=1.118; d=1.118; th=0.3;
% static gain, time constant, damping factor, time delay
if (d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else
    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end
if ((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if ((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if ((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.25-0.88*(th/t-2.8)^2))*t;
end
tdd=t/(-1.9+1.576*(th/t)^(-0.53)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));
fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f \n', kcs, tis, tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', kcd, tid, tdd);

```

<pre> pid_itae2_ex1.m kc=kcs; ti=tis; td=tds; %kc=kcd; ti=tid+0.05; td=tdd; t=0.0; t_final=15.0; x=[0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; </pre>	<pre> g_pid_itae2_ex1.m function [next_x y] = g_pid_itae2_ex1(x,delt,u) subdelt=delt/5; n=round(delt/subdelt); </pre>
---	---

Table 5.8 (Continued)

<pre> dis=0.0; delta_t=0.02; n=round(t_final/delta_t); h_u=zeros(1,500); s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array (i)=ys; % setpoint change simulation if (t>1) ys=1.0; else ys=0.0; end % disturbance rection simulation % if (t>1) dis=1.0; else dis=0.0; end s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb- yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:499 h_u(j)=h_u(j+1); end h_u(500)=u+dis; [x y]=g_pid_itae2_ex1(x,delta_t, h_u); t=t+delta_t; end figure(3); plot(t_array,ys_array,t_array, y_ar- ray); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> A=[0 -2/2.5 ; 1 -5/2.5]; B=[1.5/2.5 ; 0]; C=[0 1]; delay=0.3; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-de- lay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end </pre> <hr/> <pre> command window >> itae2_tune_ex1 kcs= 5.656, tis= 2.593, tds= 0.538 kcd=13.552, tid= 0.912, tdd= 0.409 >> pid_itae2_ex1 </pre>
---	--

5.7 Optimal Gain Margin Tuning Rule for an Unstable Second-Order Plus Time-Delay Model (OGM-unstable)

A PID controller has a structural limitation in controlling an unstable process. Thus, a PID controller with an internal feedback loop of the PD controller is recommended, as shown in Figure 5.14. For details, refer to the Section 5.8 and Chapter 7.

In this section, the optimal gain margin tuning rule for the PD controller $G_c(s) = k_i(1 + \tau_{di}s)$ is introduced for the following unstable SOPTD model:

$$G_m(s) = \frac{k \exp(-\theta s)}{(\tau s - 1)(\tau_s s + 1)} \quad (5.4)$$

Kwak *et al.* (2000) obtained the optimal tuning parameters for extensive cases of the unstable SOPTD model. And they fitted the optimal data sets obtained as shown in Table 5.9. The OGM-unstable tuning rule is applicable to all the cases of $0 < \theta < \tau - \tau_s$.

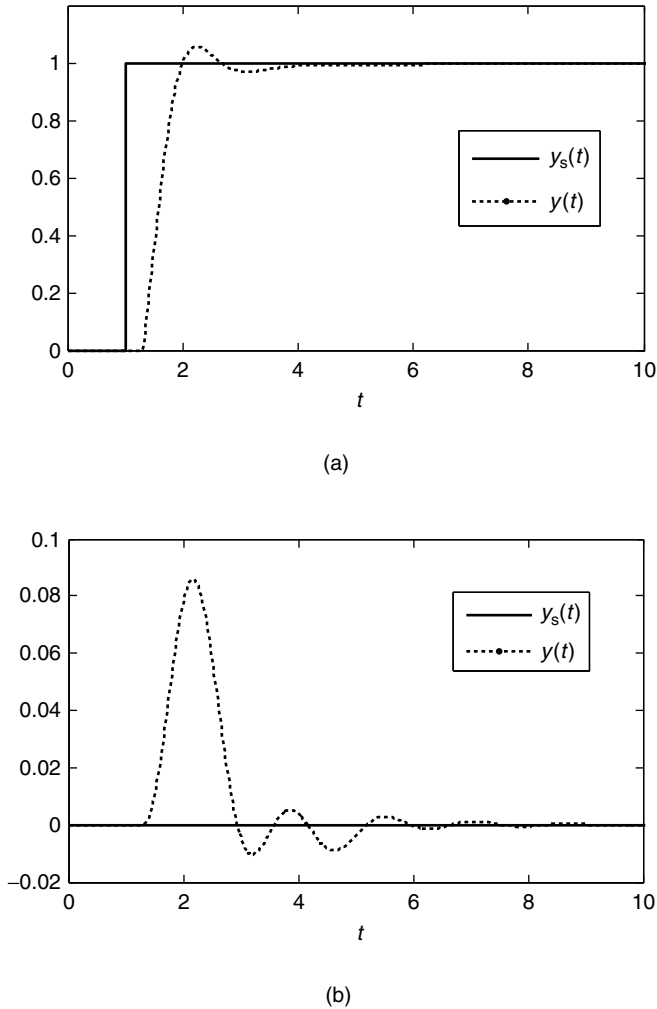


Figure 5.13 Control performances of PID controllers tuned by the ITAE-2 tuning rule: (a) step setpoint change tracking; (b) step input disturbance rejection.

Example 5.6

Obtain the tuning parameters of a PD controller for the process of which the transfer function is $G(s) = 1.0 \exp(-0.2s)/(3.0s - 1)(s + 1)$ using the OGM-unstable tuning rule.

Solution $k_c = 7.162$ and $\tau_d = 0.300$ are obtained from Table 5.10. Figure 5.15 shows the control performances of the OGM-unstable tuning rule.

5.8 Model Reduction Method for Proportional–Integral–Derivative Controller Tuning

If the model is given in the form of a high-order plus time-delay model or frequency response data, then most PID tuning rules cannot be used. In this case, the high-order model or the

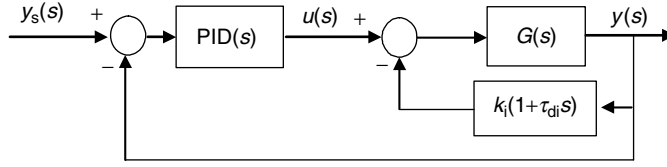


Figure 5.14 PID control combined with an internal feedback loop to control an unstable process.

Table 5.9 OGM-unstable tuning rule for an unstable SOPTD model.

Controller	Tuning parameters
k_i	τ_{di}/τ
P	$k_i = \frac{1}{\sqrt{ G_m(i\omega_u) } G_m(0) }}$
PD	$k_i = \frac{1}{\sqrt{ G_m(i\omega_u)(1 + i\tau_{di}\omega_u) } G_m(0) }} \frac{\tau_{di}}{\tau} = X_1 + X_2 \left(\frac{\theta}{\tau}\right) + X_3 \left(\frac{\theta}{\tau}\right)^2$ $X_1 = -0.003 + 0.6482 \left(\frac{\tau_s}{\tau}\right) - 2.2841 \left(\frac{\tau_s}{\tau}\right)^2 + 2.6221 \left(\frac{\tau_s}{\tau}\right)^3$ $- 0.9611 \left(\frac{\tau_s}{\tau}\right)^4$ $X_2 = 0.2446 - 1.0410 \left(\frac{\tau_s}{\tau}\right) + 13.6723 \left(\frac{\tau_s}{\tau}\right)^2 - 16.7622 \left(\frac{\tau_s}{\tau}\right)^3$ $+ 5.1471 \left(\frac{\tau_s}{\tau}\right)^4$ $X_3 = 0.1685 + 0.8289 \left(\frac{\tau_s}{\tau}\right) - 9.3630 \left(\frac{\tau_s}{\tau}\right)^2 + 2.9855 \left(\frac{\tau_s}{\tau}\right)^3$ $+ 7.3803 \left(\frac{\tau_s}{\tau}\right)^4$

frequency response data should be reduced to the FOPTD model or the SOPTD model (Sung and Lee, 1996). In this section, the model reduction method for the stable process is discussed, followed by modifications to consider the effects of the zeroes and the reduction method for the unstable process.

5.8.1 Model Reduction for Stable Processes

Assume the use of a high-order model for which the transfer function is $G_{\text{high}}(s)$ and there is a need to reduce it to the SOPTD model:

$$G_{\text{high}}(s) \cong \frac{k \exp(-\theta s)}{\tau^2 s^2 + 2\tau\xi s + 1} \quad (5.5)$$

The model reduction method fits the SOPTD model to the frequency responses of the high-order plus time-delay model. It first estimates the gain of the reduced model to fit the zero frequency response data as follows:

$$k = G_{\text{high}}(0) \quad (5.6)$$

Table 5.10 MATLAB code to simulate a PID control system tuned by the OGM-unstable tuning rule.

```

                                OGM_unstable_tune_ex1.m

clear;
k=1.0; t=3.0; ts=1.0; th=0.2; % static gain, time constant (unstable), time
constant (stable), time delay
w=0.0; delta_w=0.05;
while(1) % search boundary in which wu exists
    w=w+delta_w; g=g_OGM_unstable_ex1(w,0);
    if(imag(g)>0.0) break; end
end
w1=w-delta_w; w2=w; % w1<wu<w2
while(1) % find wu using the bisection method
    w=(w1+w2)/2; g1=g_OGM_unstable_ex1(w1,0); g=g_OGM_unstable_ex1(w,0);
    if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end
    if(abs(imag(g))<0.00000001) break; end
end
wu=w; % ultimate frequency wu is found
gu=g_OGM_unstable_ex1(wu,0);
P_ki=1/sqrt(abs(gu)*abs(k));
% tuning parameter for P controller

% - - - - -
X1=-0.003+0.6482*(ts/t)-2.2841*(ts/t)^2+2.6221*(ts/t)^3-0.9611
    *(ts/t)^4;
X2=0.2446-1.0410*(ts/t)+13.6723*(ts/t)^2-16.7622*(ts/t)^3+5.1471*
    (ts/t)^4;
X3=0.1685+0.8289*(ts/t)-9.3630*(ts/t)^2+2.9855*(ts/t)^3+7.3803
    *(ts/t)^4;
PD_tdi=t*(X1+X2*(th/t)+X3*(th/t)^2); % td for PD controller

w=0.0; delta_w=0.05;
while(1) % search boundary in which wu exists
    w=w+delta_w; g=g_OGM_unstable_ex1(w,PD_tdi);
    if(imag(g)>0.0) break; end
end
w1=w-delta_w; w2=w; % w1<wu<w2
while(1) % find wu using the bisection method
    w=(w1+w2)/2; g1=g_OGM_unstable_ex1(w1,PD_tdi); g=g_OGM_unstable_ex1
    (w,PD_tdi);
    if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end
    if(abs(imag(g))<0.00000001) break; end
end
wu=w; % ultimate frequency wu is found
gu=g_OGM_unstable_ex1(wu,PD_tdi);
g0=g_OGM_unstable_ex1(0,PD_tdi);
PD_ki=1/sqrt(abs(gu)*abs(g0)); % ki for PD controller

fprintf('P: P_ki=%6.3f \n', P_ki);
fprintf('PD: PD_ki=%6.3f, PD_tdi=%6.3f \n', PD_ki, PD_tdi);

```

Table 5.10 (Continued)

<pre> pid_OGM_unstable_ex1.m % OGM_unstable tuning parameters kc=PD_ki; td=PD_tdi; % OGM_unstable tuning parameters %kc=P_ki; td=0; t=0.0; t_final=10.0; x=[0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0; delta_t=0.01; n=round(t_final/delta_t); h_u=zeros(1,500); for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; % setpoint change simulation if(t>1) ys=1.0; else ys=0.0; end % disturbance rection simulation % if(t>1) dis=1.0; else dis=0.0; end u=kc*(ys-y)+kc*td*((ys-y)-(ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:499 h_u(j)=h_u(j+1); end h_u(500)=u+dis; [x y]=g_pid_OGM_unstable_ex1(x, delta_t,h_u); t=t+delta_t; end figure(3); plot(t_array,ys_array,t_array, y_array); legend('y_{s}(t)', 'y(t)'); figure(2); plot(t_array,u_array); </pre>	<pre> g_OGM_unstable_ex1.m function [G]=g_OGM_unstable_ex1 (w,tdi) s=i*w; G=exp(-0.2*s)*(1+tdi*s)/(3*s-1)/(s+1); end g_pid_OGM_unstable_ex1.m function [next_x y] = g_pid_OGM_unstable_ex1(x,delt,u) subdelt=delt/5; n=round(delt/subdelt); A=[0 1.0/3.0; 1.0 -2.0/3.0]; B=[2.0/3.0; 0]; C=[0 1]; delay=0.1; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end command window >> OGM_unstable_tune_ex1 P: P_ki= 3.193 PD: PD_ki= 7.162, PD_tdi= 0.300 >> pid_OGM_unstable_ex1 </pre>
---	---

It then estimates τ and ξ to satisfy the equality of (5.7) by solving (5.8) using the least-squares method. Equation (5.8) is derived from (5.7) in a straightforward manner.

$$|G_{\text{high}}(i\omega)| \approx \left| \frac{k \exp(-i\theta\omega)}{1 - \tau^2\omega^2 + i2\tau\xi\omega} \right| = \frac{k}{\sqrt{(1 - \tau^2\omega^2)^2 + (2\tau\xi\omega)^2}} \quad (5.7)$$

$$\tau^4 |G_{\text{high}}(i\omega_j)|^2 \omega_j^4 + (4\tau^2\xi^2 - 2\tau^2) |G_{\text{high}}(i\omega_j)|^2 \omega_j^2 = k^2 - |G_{\text{high}}(i\omega_j)|^2 \quad (5.8)$$

$$0 < \omega_1 < \omega_2 < \cdots < \omega_j < \cdots < \omega_n \quad (5.9)$$

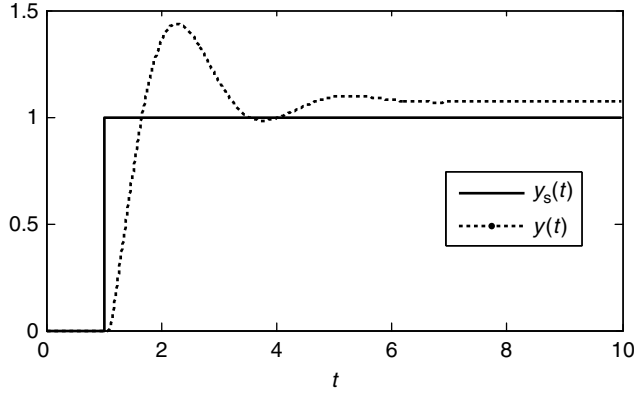


Figure 5.15 Control performances of a PD controller tuned by the OGM-unstable tuning rule.

where it is recommended to choose ω_n as the ultimate frequency ω_u of the process. If ω_u is not available, then ω_n should be chosen as the closest one to ω_u . The model reduction method finally estimates the time delay of the SOPTD model from the following phase-angle equation (5.10) with respect to ω_m . Equation (5.11) is directly obtained from (5.10).

$$\angle G_{\text{high}}(i\omega_m) = \angle \frac{k \exp(-i\theta\omega_m)}{1 - \tau^2\omega_m^2 + i2\tau\xi\omega_m} \Rightarrow \phi(\omega_m) = -\theta\omega_m + \arctan 2(-2\xi\omega_m\tau, 1 - \omega_m^2\tau^2) \quad (5.10)$$

$$\theta = \frac{-\phi(\omega_m) + \arctan 2(-2\xi\tau\omega_m, 1 - \omega_m^2\tau^2)}{\omega_m} \quad (5.11)$$

where $\arctan 2(\text{Im}, \text{Re})$ returns the phase angle of the complex number of $\text{Re} + i\text{Im}$. It returns ϕ ranged from $-\pi$ to π while $\arctan(\text{Im}/\text{Re})$ returns ϕ ranged from $-\pi/2$ to $\pi/2$. $\phi(\omega_m)$ is the phase angle of $G_{\text{high}}(s)$ at ω_m . ω_m should be $\omega_m \leq \omega_u$. It is recommended to choose ω_m as a frequency close to the ultimate frequency ω_u of the process. If $\omega_m = \omega_u$ is chosen, then (5.10) and (5.11) become the following equations:

$$-\pi = -\theta\omega_u + \arctan 2(-2\xi\omega_u\tau, 1 - \omega_u^2\tau^2) \quad (5.12)$$

$$\theta = \frac{\pi + \arctan 2(-2\xi\tau\omega_u, 1 - \omega_u^2\tau^2)}{\omega_u} \quad (5.13)$$

In summary, the reduced SOPTD model of (5.5) can be estimated from (5.6), (5.8), and (5.11). The same approach of (5.6), (5.8), and (5.11) can also be applied to the case that the frequency response data are available instead of the transfer function.

Similarly, (5.14)–(5.16) can be derived to reduce the high-order model to the FOPTD model.

$$k = G_{\text{high}}(0) \quad (5.14)$$

$$\tau = \frac{\sqrt{k^2 - |G_{\text{high}}(i\omega_m)|^2}}{|G_{\text{high}}(i\omega_m)|\omega_m} \quad (5.15)$$

$$\theta = \frac{-\phi(\omega_m) + \arctan(-\tau\omega_m)}{\omega_m} \quad (5.16)$$

5.8.2 Modification to Consider Effects of Zeroes

The above-mentioned model reduction method to obtain the reduced SOPTD model shows good performances if $G_{\text{high}}(s)$ has no zeroes or the magnitudes of the zeroes are large enough. It should be noted that the reduced SOPTD model (5.5) has no zeroes, so that it has a structural limitation in fitting the effects of the zeroes of $G_{\text{high}}(s)$. If $G_{\text{high}}(s)$ has some zeroes of which the magnitudes are small, then the least-squares method using (5.7)–(5.9) may provide the wrong parameters τ and ξ . This is because the least-squares method tries to fit the effects of the zeroes of $G_{\text{high}}(s)$ by adjusting the poles (equivalently τ and ξ) of the reduced SOPTD model. Sometimes, even though the high-order model is stable, the reduced model obtained could be unstable because the dominant zeroes of the high-order model can affect the pole positions of the reduced model. So, a modification is required to prevent the model reduction method from adjusting the poles of the reduced model to fit the zeroes of $G_{\text{high}}(s)$.

The following modification using the equivalent time-delay concept is recommended. Consider the following zeroes part of $G_{\text{high}}(s)$:

$$(-z_1^{-1}s + 1)(-z_2^{-1}s + 1) \cdots (-z_n^{-1}s + 1) = 1 - (z_1^{-1} + z_2^{-1} + \cdots + z_n^{-1})s + \cdots \quad (5.17)$$

The equivalent time delay can be represented by the Taylor series expansion:

$$\exp(-\theta_{\text{equivalent}}s) = 1 - \theta_{\text{equivalent}}s + \cdots \quad (5.18)$$

From the comparison of (5.17) and (5.18) up to the second term, the following equivalent time delay is defined to approximate the zeroes of $G_{\text{high}}(s)$:

$$\theta_{\text{equivalent}} = z_1^{-1} + z_2^{-1} + \cdots + z_n^{-1} \quad (5.19)$$

In summary, after removing the dominant stable and unstable zeroes of $G_{\text{high}}(s)$ and rearranging $G_{\text{high}}(s)$ to the modified high-order plus time-delay model of $G_{\text{high,modified}}(s)$, which has the equivalent time delay instead of the zeroes, the model reduction can be continued with $G_{\text{high,modified}}(s)$. If a more conservative approach is preferred, then it is recommended to consider only unstable zeroes to estimate the equivalent time delay and just neglect the stable zeroes because the unstable zeroes lag the phase angle and the stable zeroes lead the phase angle.

5.8.3 Model Reduction for Unstable Processes

In the same way, the model reduction method can be applied to an unstable process. Assume that the requirement is to reduce the unstable high-order process to the following unstable SOPTD model:

$$G_m(s) = \frac{k \exp(-\theta s)}{(\tau s - 1)(\tau_s s + 1)} \quad (5.20)$$

First, estimate the gain of the reduced model to fit the zero frequency response data as follows:

$$k = G_{\text{high}}(0) \quad (5.21)$$

And the model reduction method estimates τ and τ_s to satisfy the equality (5.22) by solving (5.23) using the least-squares method. Equation (5.23) is derived from (5.22) in a straightforward manner.

$$|G_{\text{high}}(i\omega)| \approx |G_m(i\omega)| = \frac{k}{\sqrt{(1 + \tau^2 \omega^2)(1 + \tau_s^2 \omega^2)}} \quad (5.22)$$

$$(\tau^2 + \tau_s^2) |G_{\text{high}}(i\omega_j)|^2 \omega_j^2 + \tau^2 \tau_s^2 |G_{\text{high}}(i\omega_j)|^2 \omega_j^4 = k^2 - |G_{\text{high}}(i\omega_j)|^2 \quad (5.23)$$

$$0 < \omega_1 < \omega_2 < \dots < \omega_j < \dots < \omega_n \quad (5.24)$$

where it is recommended to choose ω_n as the ultimate frequency ω_u of the process. If ω_u is not available, then ω_n should be chosen as the closest one to ω_u . The model reduction method finally estimates the time delay of the SOPTD model from the following phase-angle equation (5.25) with respect to ω_m . Equation (5.26) is obtained directly from (5.25).

$$\angle G_{\text{high}}(i\omega_m) = \angle G_m(i\omega_m) \Rightarrow \phi(\omega_m) = -\pi - \theta \omega_m + \arctan(\tau \omega_m) - \arctan(\tau_s \omega_m) \quad (5.25)$$

$$\theta = \frac{-\phi(\omega_m) - \pi + \arctan(\tau \omega_m) - \arctan(\tau_s \omega_m)}{\omega_m} \quad (5.26)$$

where $\phi(\omega_m)$ is the phase angle of $G_{\text{high}}(s)$ at ω_m . ω_m should be $\omega_m \leq \omega_u$. It is recommended to choose ω_m as a frequency close to the ultimate frequency ω_u of the process. If $\omega_m = \omega_u$ is chosen, then (5.25) and (5.26) become the following equations:

$$-\pi = -\pi - \theta \omega_u + \arctan(\tau \omega_u) - \arctan(\tau_s \omega_u) \quad (5.27)$$

$$\theta = \frac{\arctan(\tau \omega_u) - \arctan(\tau_s \omega_u)}{\omega_u} \quad (5.28)$$

In summary, the reduced unstable SOPTD model (5.20) can be estimated from (5.21), (5.23) and (5.26).

Example 5.7

Obtain the tuning parameters of a PID controller for the process of which the transfer function is $G_{\text{high}}(s) = \exp(-0.1s)/(s + 1)^3$ using the IMC tuning rule.

Solution $k = 1.0$ is obtained from $G_{\text{high}}(0) = 1.0$. Because $G_{\text{high}}(i1.5) = -0.1705 - i0.0074$, $\phi(1.5) = \arctan 2(-0.0074, -0.1705) = -3.0984$ and $|G_{\text{high}}(i1.5)| = 0.1707$ at $\omega_m = 1.5$. From (5.15), $\tau = 3.849$ is obtained. Also, $\theta = 1.133$ is obtained from (5.16). So, the reduced FOPTD model obtained is $G_m(s) = \exp(-1.133s)/(3.849s + 1)$. Then, the IMC tuning rule provides $k_c = 3.118$, $\tau_i = 4.415$ and $\tau_d = 0.494$ from Table 5.3. The MATLAB code is shown in Table 5.11.

Table 5.11 MATLAB code to obtain the reduced FOPTD model in Example 5.7.

<pre> mr_ex1.m clear; w=1.5; g=g_mr_ex1(w); k=abs(g_mr_ex1(0)); tau=sqrt(k^2-abs(g)^2)/abs(g)/w; theta=(-atan2(imag(g),real(g))+atan(-tau*w))/w; % k: static gain, tau: time constant, theta: time delay fprintf('k=%5.3f tau=%5.3f theta=%5.3f\n',k,tau,theta); </pre>	<pre> g_mr_ex1.m function [G]=g_mr_ex1(w) s=i*w; G=exp(-0.1*s)/(s+1)^3; end </pre> <hr/> <pre> command widow >> mr_ex1 k=1.000 tau=3.849 theta=1.133 >> imc_ex1 PI: kcs= 2.293, tis= 4.415 PID: kcs= 3.118, tis= 4.415, tds= 0.494 </pre>
<pre> imc_ex1.m k=k; t=tau; th=theta; r=1.7*th; pi_kcs=(2*t+th)/2/r/k; pi_tis=t+th/2; r=0.25*th; pid_kcs=(2*t+th)/2/(r+th)/k; pid_tis=t+th/2; pid_tds=t*th/(2*t+th); fprintf('PI: kcs=%6.3f, tis=%6.3f\n',pi_kcs,pi_tis); fprintf('PID: kcs=%6.3f, tis=%6.3f, tds=%6.3f\n',pid_kcs,pid_tis,pid_tds); </pre>	

Example 5.8

Obtain the tuning parameters of the PID controller for the process of which the transfer function is $G(s) = \exp(-0.1s)/(s + 1)^3$ using the ITAE-2 tuning rule.

Solution $k = 1.0$ is obtained from $G_{\text{high}}(0) = 1.0$. In (5.8), let us define $\hat{p}_1 = \tau^4$, $\varphi_{1,k} = |G_{\text{high}}(i\omega_k)|^2 \omega_k^4$, $\hat{p}_2 = 4\tau^2 \xi^2 - 2\tau^2$, $\varphi_{2,k} = |G_{\text{high}}(i\omega_k)|^2 \omega_k^2$ and $y_k = k - |G_{\text{high}}(i\omega_k)|^2$. Then, \hat{p}_1 and \hat{p}_2 can be estimated by the least-squares method mentioned in Chapter 2. Then, $\tau = \hat{p}_1^{1/4}$ and $\xi = \sqrt{(\hat{p}_2 + 2\tau^2)/(4\tau^2)}$ are obtained. And θ is obtained from (5.13). The source code for the model reduction method is shown in Table 5.12. The estimated SOPTD model is $G_{r-m}(s) = 1.000 \exp(-0.477s)/(1.510^2 s^2 + 2 \times 1.510 \times 0.861s + 1)$. Then, $k_c = 2.784$, $\tau_i = 2.702$ and $\tau_d = 0.924$ for the step setpoint change and $k_c = 6.866$, $\tau_i = 1.382$ and $\tau_d = 0.685$ for the step input disturbance rejection are obtained from Table 5.7.

Table 5.12 MATLAB code to obtain the reduced SOPTD model in Example 5.8.

mr_ex2.m	g_mr_ex2.m
<pre> clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_mr_ex2(w); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_mr_ex2(w1); g=g_mr_ex2(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found k=abs(g_mr_ex2(0)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_mr_ex2(w); y(j,1)=k^2-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^4; phi_2(j,1)=(abs(G(j))^2)*w^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv(PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/(4*tau^2)) ^0.5; theta=(pi+atan2(-2*xi*tau*wu,1-wu^2*- tau^2))/wu; % tau: time constant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n',k,tau); fprintf('xi=%5.3f theta=%5.3f \n',xi, theta); </pre>	<pre> function [G]=g_mr_ex2(w) s=i*w; G=exp(-0.1*s)/(s+1)^3; end </pre> <hr/> <pre> command window >> mr_ex2 k=1.000 tau=1.510 xi=0.861 theta=0.477 >> itae_ex2 kcs= 2.784, tis= 2.702, tds= 0.924 kcd= 6.866, tid= 1.382, tdd= 0.685 </pre>

itae_ex2.m

```

k=k; t=tau; d=xi; th=theta;
% static gain, time constant, damping factor, time delay
if(d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else

```

Table 5.12 (Continued)

```

    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end
if((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));

fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f \n', kcs, tis, tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', kcd, tid, tdd);

```

Example 5.9

Obtain the tuning parameters of a PID controller using the ITAE-2 tuning rule for the process of which the frequency responses are as follows: $G_{\text{high}}(i0.0) = 1.00 - i0.00$, $G_{\text{high}}(i0.2) = 0.50 - i0.76$, $G_{\text{high}}(i0.4) = -0.22 - i0.65$, $G_{\text{high}}(i0.6) = -0.42 - i0.20$, $G_{\text{high}}(i0.8) = -0.28 - i0.07$, $G_{\text{high}}(i1.0) = -0.13 + i0.13$.

Solution $k = 1.0$ is obtained from $G_{\text{high}}(0) = 1.0$. In (5.8), let us define $\hat{p}_1 = \tau^4$, $\varphi_{1,k} = |G_{\text{high}}(i\omega_k)|^2 \omega_k^4$, $\hat{p}_2 = 4\tau^2 \xi^2 - 2\tau^2$, $\varphi_{2,k} = |G_{\text{high}}(i\omega_k)|^2 \omega_k^2$ and $y_k = 1 - |G_{\text{high}}(i\omega_k)|^2$. Then, \hat{p}_1 and \hat{p}_2 can be estimated by the least-squares method mentioned in Chapter 2. Then, $\tau = \hat{p}_1^{1/4}$ and $\xi = \sqrt{(\hat{p}_2 + 2\tau^2)/(4\tau^2)}$ are obtained. And θ is obtained from (5.11). It should be noted that the sixth frequency data of $G_{\text{high}}(i1.0) = -0.13 + i0.13$ should not be used to estimate the time delay in (5.11) because it is out of the range of $\arctan 2(\text{Im}, \text{Re})$. So, the fifth frequency response data of $G_{\text{high}}(i0.8) = -0.28 - i0.07$ is used in (5.11) in this case. The source code for the least-squares method is shown in Table 5.13. The estimated SOPTD model is $G_{r-m}(s) = 1.000 \exp(-0.852s)/(2.205^2 s^2 + 2 \times 2.205 \times 0.797s + 1)$. Then, $k_c = 2.151$, $\tau_i = 3.658$ and $\tau_d = 1.465$ for the step setpoint change and $k_c = 4.932$, $\tau_i = 2.313$ and $\tau_d = 1.173$ for the step input disturbance rejection change are obtained from Table 5.13.

Example 5.10

Obtain the tuning parameters of a PID controller using the ITAE-2 tuning rule for the following process:

$$G_{\text{high}}(s) = \frac{(-0.3s + 1)\exp(-0.3s)}{(s + 1)^3} \quad (5.29)$$

Solution The unstable zero part of $(-0.3s + 1)$ is approximated by the equivalent time delay of $\exp(-0.3s)$. Then, process (5.29) can be approximated by the following modified model:

$$G_{\text{high,modified}}(s) = \frac{\exp(-0.6s)}{(s + 1)^3} \quad (5.30)$$

Now, (5.30) can be reduced to the SOPTD model $G_m(s) = 1.000 \exp(-1.004s)/(1.440^2 s^2 + 2 \times 1.440 \times 0.908s + 1)$ by using the model reduction method in Table 5.14. Then, the tuning parameters $k_c = 1.396$, $\tau_i = 2.752$ and $\tau_d = 0.929$ are obtained by the ITAE-2-setpoint tuning

Table 5.13 MATLAB code to obtain the reduced SOPTD model in Example 5.9.

mr_ex3.m	command window
<pre>clear; w(1)=0.0; w(2)=0.2; w(3)=0.4; w(4) =0.6; w(5)=0.8; w(6)=1.0; G(1)=1.0+i*0.0; G(2)=0.5-i*0.76; G (3)=-0.22-i*0.65; G(4)=-0.42-i*0.2; G(5)=-0.28-i*0.07; G(6)=-0.13+i*0.13; k=abs(G(1)); for j=1:6 % least square method y(j,1)=k^2-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w(j)^4; phi_2(j,1)=(abs(G(j))^2)*w(j)^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv (PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat (2)+2*tau^2)/(4*tau^2))^0.5; PA=atan2(imag(G(5)),real(G(5))); theta=(-PA+atan2(-2*xi*tau*w(5),1-w (5)^2*tau^2))/w(5); % tau: time constant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n', k, tau); fprintf('xi=%5.3f theta=%5.3f \n', xi, theta);</pre>	<pre>>> mr_ex3 k=1.000 tau=2.205 xi=0.797 theta=0.852 >> itae_ex3 kcs= 2.150, tis= 3.658, tds= 1.465 kcd= 4.932, tid= 2.313, tdd= 1.173</pre>

Table 5.13 (Continued)

```

                                itae_ex3.m
k=k; t=tau; d=xi; th=theta;
% static gain, time constant, damping factor, time delay
if(d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else
    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end

if((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)

    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));

fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f \n', kcs, tis, tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', kcd, tid, tdd);

```

rule. Figure 5.16 shows the control performance. The ITAE-2 tuning rule on the basis of the approximation model (5.30) shows excellent tuning results. It confirms that the approximation using the equivalent time delay is acceptable.

Example 5.11

Obtain the tuning parameters of a PID controller using the ITAE-2-setpoint tuning rule for the following process:

$$G_{\text{high}}(s) = \frac{(0.7s + 1)(-0.3 + 1)\exp(-0.4s)}{(s + 1)^3} \quad (5.31)$$

Solution In this case, the stable zero part of $(0.7s + 1)$ should be eliminated for a conservative model reduction. And the unstable zero part of $(-0.3 + 1)$ is approximated by

Table 5.14 MATLAB code to solve the design problem and simulate the PID controller in Example 5.10.

<pre> mr_ex4.m clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_mr_ex4(w); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method </pre>	<pre> command window >> mr_ex4 k=1.000 tau=1.440 xi=0.908 theta=1.004 >> itae_ex4 kcs= 1.396, tis= 2.752, tds= 0.929 kcd= 2.560, tid= 2.099, tdd= 0.951 >> pid_itae2_ex4 </pre>
<pre> w=(w1+w2)/2; g1=g_mr_ex4(w1); g=g_mr_ex4(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end </pre>	<pre> g_mr_ex4.m function [G]=g_mr_ex4(w) s=i*w; G=exp(-0.6*s)/(s+1)^3; end </pre>
<pre> wu=w; % ultimate frequency wu is found k=abs(g_mr_ex4(0)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_mr_ex4(w); y(j,1)=k^2-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^4; phi_2(j,1)=(abs(G(j))^2)*w^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv (PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/(4*tau^2)) ^0.5; theta=(pi+atan2(-2*xi*tau*wu,1- wu^2*tau^2))/wu; % tau: time constant, xi: damping fac- tor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n', k, tau); fprintf('xi=%5.3f theta=%5.3f \n', xi, theta); </pre>	<pre> g_pid_itae2_ex4.m function [next_x y]=g_pid_ itae2_ex4(x,delt,u) subdelt=delt/5; n=round(delt/subdelt); A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 ; -0.3; 0]; C=[0 0 1]; delay=0.3; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end </pre>
<pre> itae_ex4.m k=k; t=tau; d=xi; th=theta; % static gain, time constant, damping factor, time delay if(d<=0.9) kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k; else </pre>	

Table 5.14 (Continued)

```

    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end
if((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));
fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f \n',kcs,tis,tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n',kcd,tid,tdd);

```

```

                                pid_itae2_ex4.m

kc=kcs; ti=tis; td=tds;
t=0.0; t_final=15.0;
x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0;
delta_t=0.02; n=round(t_final/delta_t);
h_u=zeros(1,500); s=0.0;
for i=1:n
    t_array(i)=t; y_array(i)=y; ys_array(i)=ys;
    if(t>1) ys=1.0; else ys=0.0; end % setpoint change simulation
%   if(t>1) dis=1.0; else dis=0.0; end % disturbance rection simulation
    s=s+(kc/ti)*(ys-y)*delta_t;
    u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t;
    ysb=ys; yb=y; % one sampling before
    u_array(i)=u;
    for j=1:499
        h_u(j)=h_u(j+1);
    end
    h_u(500)=u+dis;
    [x y]=g_pid_itae2_ex4(x,delta_t,h_u);
    t=t+delta_t;
end
figure(3); plot(t_array,ys_array,t_array,y_array); legend('y_{s}
(t)','y(t)');
figure(2); plot(t_array,u_array);

```

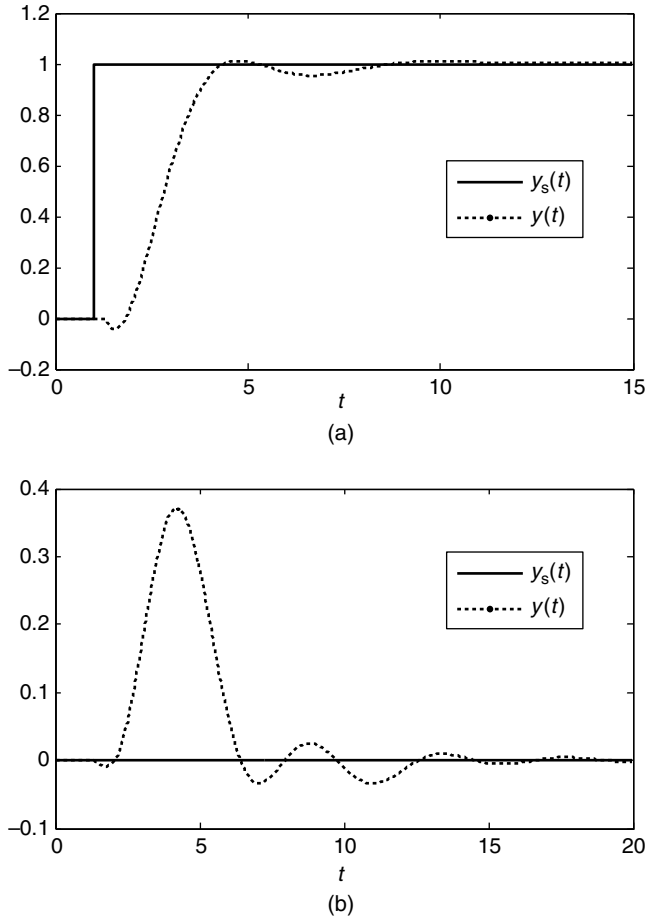


Figure 5.16 Tuning results of the ITAE-2 tuning rule with the approximated model in Example 5.10.

the equivalent time delay of $\exp(-0.3s)$. Then, the modified model is

$$G_{\text{high,modified}}(s) = \frac{\exp(-0.7s)}{(s+1)^3} \quad (5.32)$$

Finally, the reduced SOPTD model (5.33) is obtained by applying the model reduction method to $G_{\text{high,modified}}(s)$.

$$G_{\text{r-m}}(s) = \frac{\exp(-1.107s)}{1.431^2 s^2 + 2 \times 1.431 \times 0.913s + 1} \quad (5.33)$$

Finally, the tuning parameters $k_c = 1.286$, $\tau_i = 2.758$ and $\tau_d = 0.937$ are obtained from Table 5.7.

Table 5.15 and Figure 5.17 show the control performance of a PID controller designed by the ITAE-2 tuning rule based on the approximate model. Because the stable zero is neglected, a conservative control action is obtained.

Table 5.15 MATLAB code to solve the design problem and simulate the PID controller in Example 5.11.

<pre> mr_ex5.m clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_mr_ex5(w); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_mr_ex5(w1); g=g_mr_ex5(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end </pre>	<pre> command window >> mr_ex5 k=1.000 tau=1.431 xi=0.913 theta=1.107 >> itae_ex5 kcs= 1.286, tis= 2.758, tds= 0.937 kcd= 2.259, tid= 2.196, tdd= 0.981 >> pid_itae2_ex5 </pre>
<pre> if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found k=abs(g_mr_ex5(0)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_mr_ex5(w); y(j,1)=k^2-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^4; phi_2(j,1)=(abs(G(j))^2)*w^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv (PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/(4*tau^2)) ^0.5; theta=(pi+atan2(-2*xi*tau*wu,1-wu^2*- tau^2))/wu; % tau: time contant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n',k,tau); fprintf('xi=%5.3f theta=%5.3f \n',xi,theta); </pre>	<pre> g_mr_ex5.m function [G]=g_mr_ex5(w) s=i*w; G=exp(-0.7*s)/(s+1)^3; end g_pid_itae2_ex5.m function [next_x y]=g_pid_ itae2_ex5(x,delt,u) subdelt=delt/5; n=round (delt/subdelt); A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1 ; 0.4; -0.21]; C=[0 0 1]; delay=0.4; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end </pre>

(continued)

Table 5.15 (Continued)

```

                                itae_ex5.m
k=k; t=tau; d=xi; th=theta;
% static gain, time constant, damping factor, time delay
if(d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else
    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end
if((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));
fprintf(' kcs=%6.3f, tis=%6.3f, tds=%6.3f \n', kcs, tis, tds);
fprintf(' kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', kcd, tid, tdd);

```

```

                                pid_itae2_ex5.m
%kc=kcs; ti=tis; td=tds;
kc=kcd; ti=tid; td=tdd;
t=0.0; t_final=20.0;
x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0;
delta_t=0.02; n=round(t_final/delta_t);
h_u=zeros(1,500); s=0.0;
for i=1:n
    t_array(i)=t; y_array(i)=y; ys_array(i)=ys;
%   if(t>1) ys=1.0; else ys=0.0; end % setpoint change simulation
    if(t>1) dis=1.0; else dis=0.0; end % disturbance rection simulation
    s=s+(kc/ti)*(ys-y)*delta_t;
    u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t;
    ysb=ys; yb=y; % one sampling before
    u_array(i)=u;
    for j=1:499
        h_u(j)=h_u(j+1);
    end
end

```

Table 5.15 (Continued)

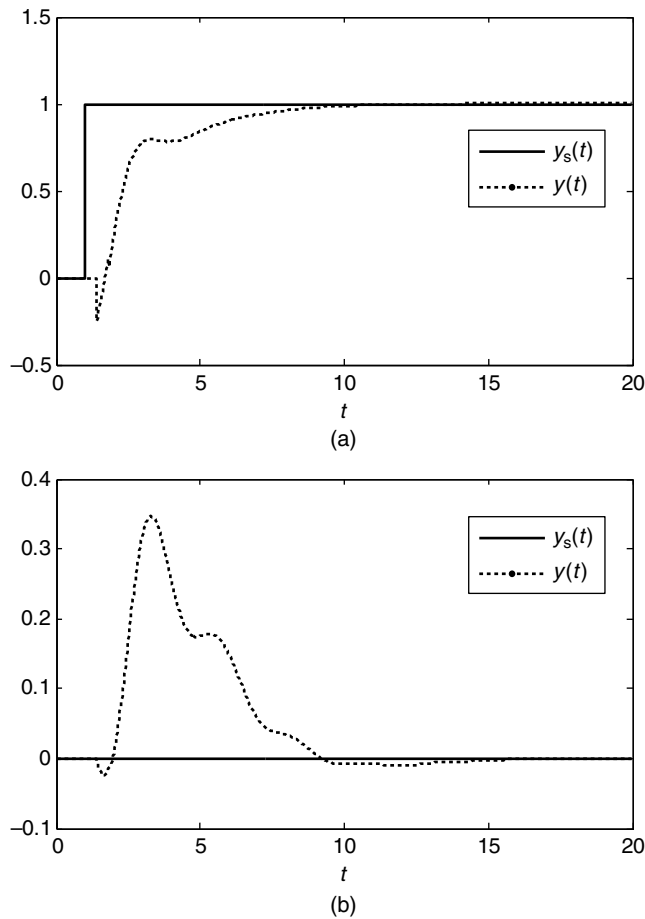
```

end
h_u(500)=u+dis;
[x y]=g_pid_itae2_ex5(x,delta_t,h_u);
t=t+delta_t;
end
figure(4); plot(t_array,ys_array,t_array,y_array); legend('y_{s}(t)', 'y(t)');
figure(2); plot(t_array,u_array);

```

Example 5.12

Design the cascade control of Figure 5.18. For a detailed description on the cascade control, refer to Chapter 7.

**Figure 5.17** Tuning results of the ITAE-2 tuning rule with the approximated model in Example 5.11.

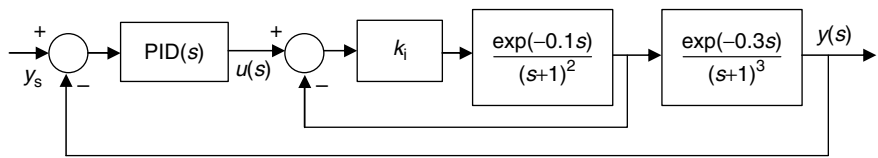


Figure 5.18 Cascade control system.

The ultimate gain of the open-loop transfer function $\exp(-0.1s)/(s + 1)^2$ is $k_u = 20.67$. Let us choose $k_i = 6.89(k_i = k_u/3)$ as the proportional gain of the internal feedback loop. Then, the transfer function from $u(t)$ to $y(t)$ is

$$G_{\text{overall}}(s) = \frac{y(s)}{u(s)} = \frac{6.89 \exp(-0.4s)/(s + 1)^5}{1 + 6.89 \exp(-0.1s)/(s + 1)^2} \tag{5.34}$$

Note that the PID controller should be tuned for the overall process. Then, (5.34) should be reduced to the SOPTD model to tune the PID controller using the ITAE-2-setpoint. The reduced SOPTD model obtained by the model reduction method is $G_m(s) = 0.873 \exp(-1.136s)/(1.261^2 s^2 + 2 \times 1.261 \times 0.964s + 1)$ and the tuning parameters by the ITAE-2-setpoint based on the reduced model are $k_c = 1.390$, $\tau_i = 2.577$ and $\tau_d = 0.823$.

The MATLAB code for Example 5.12 and the simulation results are shown in Table 5.16 and Figure 5.19 respectively.

Example 5.13

Consider the PID control combined with the internal feedback loop of the P controller in Figure 5.20. Design the PID controller and the internal P controller of $G_{ci}(s) = k_i$ for $G(s) = \exp(-0.2s)/[(5s - 1)(s + 1)^2]$.

Solution First, the unstable process $G(s) = \exp(-0.2s)/[(5s - 1)(s + 1)^2]$ is reduced to the SOPTD model $G_m(s) = \exp(-0.713s)/(4.860s - 1)/(1.616s + 1)$ by the model reduction method for the unstable process. Then, the P controller tuned by the optimal gain margin

Table 5.16 MATLAB code to solve the design problem and simulate the PID controller in Example 5.12.

<pre>mr_ex6.m clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_mr_ex6(w); if (imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1<wu<w2 while(1) % find wu using the bisection method</pre>	<pre>command window >> mr_ex6 k=0.873 tau=1.261 xi=0.964 theta=1.136 >> itae_ex6 kcs= 1.390, tis= 2.577, tds= 0.823 kcd= 2.273, tid= 2.108, tdd= 0.880 >> pid_itae2_ex6 g_mr_ex6.m function [G]=g_mr_ex6(w) s=i*w;</pre>
--	---

Table 5.16 (Continued)

<pre> w=(w1+w2)/2; g1=g_mr_ex6(w1); g=g_mr_ex6(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found k=abs(g_mr_to_second(0)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_mr_ex6 (w); y(j,1)=k-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^4; phi_2(j,1)=(abs(G(j))^2)*w^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv (PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi= ((P_hat(2)+2*tau^2)/(4*tau^2)) ^0.5; theta=(pi+atan2(-2*xi*tau*wu,1- wu^2*tau^2))/wu; % tau: time constant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n',k, tau); fprintf('xi=%5.3f theta=%5.3f \n', xi,theta); </pre>	<pre> G=(6.89*exp(-0.4*s)/(s+1)^5)/(1 +6.89*exp(-0.1*s)/(s+1)^2); end </pre>
<pre> g_pid_itae2_ex6_p.m function [next_x,y]=g_pid_itae2_ex6_p(x, delt,u); subdelt=delt/5; n=round(delt/sub- delt); A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1; 0; 0]; C=[0 0 1]; delay=0.3; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return </pre>	<pre> g_pid_itae2_ex6_s.m function [next_x,y]=g_pid_i- tae2_ex6_s(x,delt,u); subdelt=delt/5; n=round(delt/sub- delt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.1; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return </pre>

Table 5.16 (Continued)

```

                                itae_ex6.m
k=k; t=tau; d=xi; th=theta;
% static gain, time constant, damping factor, time delay
if(d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else
    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832)*d)/k;
end
if((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else
    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));
fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f \n', kcs, tis, tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f \n', kcd, tid, tdd);

```

```

                                pid_itae2_ex6.m
kc1=kcs; til=tis; tdl=tds; %primary PID
kc2=6.89; %secondary P
ys1=1.0; dis=0.0; %setpoint and disturbance
tf=20; delt=0.02; tf_k=round(tf/delt);
uu1=zeros(1,500); uu2=zeros(1,500); yy2=zeros(1,500);
x1=zeros(3,1); x2=zeros(2,1);
y1=0.0; y1b=0.0; s1=0.0; ys1b=0.0; y2=0.0; y2b=0.0;
for k=1:tf_k
    t=(k-1)*delt;
    T(k)=t; Y1(k)=y1; Y2(k)=y2; Ys1(k)=ys1;
    U1(k)=uu1(500); U2(k)=uu2(500);
    for i=1:499 uu1(i)=uu1(i+1); end
    for i=1:499 uu2(i)=uu2(i+1); end
    for i=1:499 yy2(i)=yy2(i+1); end
    s1=s1+(kc1/til)*(ys1-y1)*delt;
    uu1(500)=kc1*(ys1-y1)+s1+kc1*tdl*(ys1-y1-ys1b+y1b)/delt;

```

Table 5.16 (Continued)

```

uu2(500)=kc2*(uu1(500)-y2);
yy2(500)=y2;
y1b=y1; ys1b=ys1;
[x2,y2]=g_pid_itae2_ex6_s(x2,delt,uu2); %cascade
y2=y2+dis; %disturbance
[x1,y1]=g_pid_itae2_ex6_p(x1,delt,yy2);
end
figure(1); plot(T,Ys1,T,Y1);
figure(2); plot(T,U1);

```

tuning rule in Table 5.9 is $k_i = 2.177$. Now, the overall transfer function from $u(t)$ to $y(t)$ of $G_{\text{overall}}(s) = G(s)/(1 + G(s)k_i)$ can be reduced to (5.35) using the model reduction method for the stable process. Then, the PID controller can be designed by the ITAE-2-setpoint on the basis of the reduced model as shown in (5.35):

$$G_m(s) = \frac{0.850 \exp(-0.792s)}{2.633^2 s^2 + 2 \times 2.633 \times 0.305s + 1}, \quad G_c(s) = 1.182 \left(1 + \frac{1}{1.666s} + 4.157s \right) \quad (5.35)$$

The MATLAB code and the simulation result are shown in Table 5.17 and Figure 5.21 respectively.

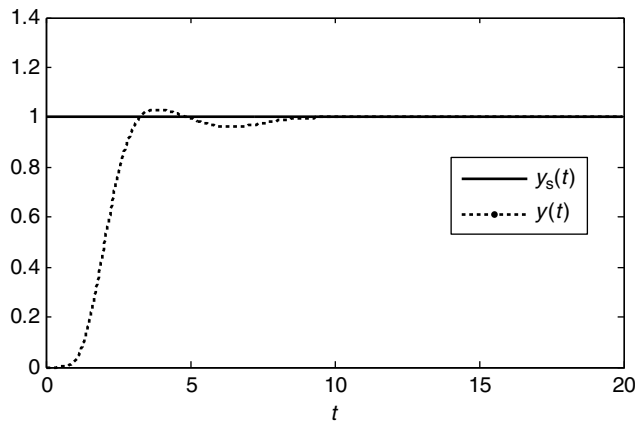
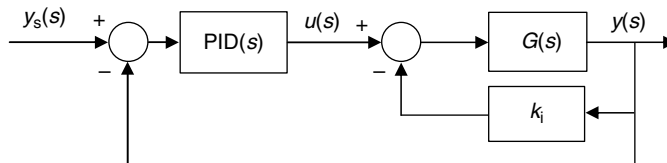
**Figure 5.19** Tuning results of the ITAE-2 rule on the basis of the reduced model in Example 5.12.**Figure 5.20** PID control combined with an internal feedback loop to control an unstable process.

Table 5.17 MATLAB code to solve the design problem and simulate the PID controller in Example 5.13.

<pre> mr_ex7.m clear; w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_mr_ex7(w); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1<wu<w2 while(1) % findwu using the bisection method w=(w1+w2)/2; g1=g_mr_ex7(w1); g=g_mr_ex7(w); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; </pre>	<pre> command window >> mr_ex7 k=1.000 tau=4.860 taus=1.616 theta=0.713 >> OGM_unstable_ex7 P: P_ki= 2.177 PD: PD_ki= 4.236, PD_tdi= 0.787 >> mr_overall_ex7 k=0.850 tau=2.633 xi=0.305 theta=0.792 >> itae_ex7 kcs= 1.182, tis= 1.666, tds= 4.157 kcd= 5.054, tid= 2.328, tdd= 1.715 >> pid_OGM_unstable_ex7 </pre>
<pre> end end wu=w; % ultimate frequency wu is found k=abs(g_mr_ex7(0)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_mr_ex7(w); y(j,1)=k^2-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^2; phi_2(j,1)=(abs(G(j))^2)*w^4; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv(PHI'*PHI)*PHI'*Y; tau2=(P_hat(1)+sqrt(P_hat(1)^2- 4*P_hat(2)))/2; tau=sqrt(tau2); taus=sqrt(P_hat(2)/tau^2); theta=(atan(tau*wu)-atan(taus*- wu))/wu; % tau: time constant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5. 3f taus=%5.3f theta=%5.3f \n',k, tau, taus, theta); </pre>	<pre> g_mr_ex7.m function [G]=g_mr_ex7(w) s=i*w; G=exp(-0.2*s)/((5*s-1)*(s+1)^2); end g_mr_overall_ex7.m function [G]=g_mr_overall_ex7(w) s=i*w; Gp=exp(-0.2*s)/(5*s-1)/(s+1)^2; Gc=2.177; G=Gp/(1+Gp*Gc); end </pre>
<pre> g_pid_OGM_unstable_ex7.m function [next_x y] = g_pid_OGM_un- stable_ex7(x,delt,u) </pre>	<pre> g_OGM_unstable_ex7.m function [G]=g_OGM_unstable_ex7(w,tdi) </pre>

Table 5.17 (Continued)

<pre> subdelt=delt/5; n=round(delt/sub- delt); A=[0 0 1/5 ; 1 0 -3/5 ; 0 1 -9/5]; B=[1/5; 0 ; 0]; C=[0 0 1]; delay=0.2; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; end </pre>	<pre> s=i*w; G=exp(-0.2*s)*(1+tdi*s)/(5*s-1)/ (s+1)^2; end </pre>
<pre> OGM_unstable_ex7.m k=k; t=tau; ts=taus; th=theta; % static gain, time constant(unstable), time constant(stable), time delay w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_OGM_unstable_ex7(w,0); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_OGM_unstable_ex7(w1,0); g=g_OGM_unstable_ex7(w,0); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.00000001) break; end end wu=w; % ultimate frequency wu is found gu=g_OGM_unstable_ex7(wu,0); P_ki=1/sqrt(abs(gu)*abs(k)); % tuning parameter for P controller %----- X1=-0.003+0.6482*(ts/t)-2.2841*(ts/t)^2+2.6221*(ts/t)^3-0.9611*(ts/ t)^4; X2=0.2446-1.0410*(ts/t)+13.6723*(ts/t)^2-16.7622*(ts/t)^3+5.1471* (ts/t)^4; X3=0.1685+0.8289*(ts/t)-9.3630*(ts/t)^2+2.9855*(ts/t)^3+7.3803*(ts/ t)^4; PD_tdi=t*(X1+X2*(th/t)+X3*(th/t)^2); % td for PD controller w=0.0; delta_w=0.05; while(1) % search boundary in which wu exists w=w+delta_w; g=g_OGM_unstable_ex7(w,PD_tdi); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_OGM_unstable_ex7(w1,PD_tdi); g=g_OGM_unstable_ex7 (w,PD_tdi); </pre>	

(continued)

Table 5.17 (Continued)

```

    if (imag(g)*imag(g1)>0.0) w1=w; else w2=w; end
    if (abs(imag(g))<0.00000001) break; end
end
wu=w; % ultimate frequency wu is found
gu=g_OGM_unstable_ex7(wu,PD_tdi);
g0=g_OGM_unstable_ex7(0,PD_tdi);
PD_ki=1/sqrt(abs(gu)*abs(g0)); % ki for PD controller
fprintf('P: P_ki=%6.3f \n',P_ki);
fprintf('PD: PD_ki=%6.3f, PD_tdi=%6.3f \n',PD_ki,PD_tdi);

```

```

                                mr_overall_ex7.m
w=0.0; delta_w=0.05;
while(1) % search boundary in which wu exists
    w=w+delta_w; g=g_mr_overall_ex7(w);
    if (imag(g)>0.0) break; end
end
w1=w-delta_w; w2=w; % w1<wu<w2
while(1) % find wu using the bisection method
    w=(w1+w2)/2; g1=g_mr_overall_ex7(w1); g=g_mr_overall_ex7(w);
    if (imag(g)*imag(g1)>0.0) w1=w; else w2=w; end
    if (abs(imag(g))<0.000001) break; end
end
wu=w; % ultimate frequency wu is found
k=abs(g_mr_overall_ex7(0));
for j=1:10 % least square method
    w=(j-1)*wu/9.0; G(j)=g_mr_overall_ex7(w);
    y(j,1)=k^2-(abs(G(j))^2);
    phi_1(j,1)=(abs(G(j))^2)*w^4;
    phi_2(j,1)=(abs(G(j))^2)*w^2;
end % P_hat: solution of the least square method
PHI=[phi_1 phi_2]; Y=y; P_hat=inv(PHI'*PHI)*PHI'*Y;
tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/(4*tau^2))^0.5;
theta=(pi+atan2(-2*xi*tau*wu,1-wu^2*tau^2))/wu;
% tau: time constant, xi: damping factor, theta: time delay
fprintf('k=%5.3f tau=%5.3f \n',k,tau);
fprintf('xi=%5.3f theta=%5.3f \n',xi,theta);

```

```

                                itae_ex7.m
k=k; t=tau; d=xi; th=theta;
% static gain, time constant, damping factor, time delay
if (d<=0.9)
    kcs=(-0.04+(0.333+0.949*(th/t)^(-0.983))*d)/k;
else
    kcs=(-0.544+0.308*th/t+1.408*(th/t)^(-0.832))*d/k;
end
if ((th/t)<=1.0)
    tis=(2.055+0.072*th/t)*d*t;
else

```

Table 5.17 (Continued)

```

    tis=(1.768+0.329*th/t)*d*t;
end
tds=t/(1.0-exp(-(th/t)^(1.060)*d/0.870))/(0.55+1.683*(th/t)^
(-1.090));
if((th/t)<0.9)
    kcd=(-0.670+0.297*(th/t)^(-2.001)+2.189*(th/t)^(-0.766)*d)/k;
else
    kcd=(-0.365+0.260*(th/t-1.400)^2+2.189*(th/t)^(-0.766)*d)/k;
end
if((th/t)<0.4)
    tid=(2.212*(th/t)^(0.520)-0.300)*t;
else
    tid=(-0.975+0.910*(th/t-1.845)^2+(1-exp(-d/(0.150+0.330*th/t)))*
(5.250-0.880*(th/t-2.800)^2))*t;
end
tdd=t/(-1.900+1.576*(th/t)^(-0.530)+(1-exp(-d/(-0.15+0.939*(th/t)^
(-1.121))))*(1.45+0.969*(th/t)^(-1.171)));
fprintf('kcs=%6.3f, tis=%6.3f, tds=%6.3f\n',kcs,tis,tds);
fprintf('kcd=%6.3f, tid=%6.3f, tdd=%6.3f\n',kcd,tid,tdd);

```

pid_OGM_unstable_ex7.m

```

%kci=PD_ki; tdi=PD_tdi; % OGM_unstable tuning parameters
kci=P_ki; tdi=0.0; % OGM_unstable tuning parameters
kc=kcs; ti=tis; td=tds; % ITAE-2-setpoint tuning parameters
t=0.0; t_final=50.0; x=[0 0 0]'; y=0.0; yb=0.0; ys=0.0; ysb=0.0; dis=0.0;
delta_t=0.02; n=round(t_final/delta_t); s=0.0; h_u=zeros(1,500);
for i=1:n
    t_array(i)=t; y_array(i)=y; ys_array(i)=ys;
    if(t>1) ys=1.0; else ys=0.0; end % setpoint change simulation
    % if(t>1) dis=1.0; else dis=0.0; end % disturbance rection simulation
    s=s+(kc/ti)*(ys-y)*delta_t;
    u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delta_t;
    ui=kci*y+kci*tdi*(y-yb)/delta_t;
    ysb=ys; yb=y; % one sampling before
    u_array(i)=u-ui;
    for j=1:499
        h_u(j)=h_u(j+1);
    end
    h_u(500)=u-ui+dis;
    [x y]=g_pid_OGM_unstable_ex7(x,delta_t,h_u);
    t=t+delta_t;
end
figure(3); plot(t_array,ys_array,t_array,y_array);
legend('y_{s}(t)', 'y(t)');
figure(2); plot(t_array,u_array);

```

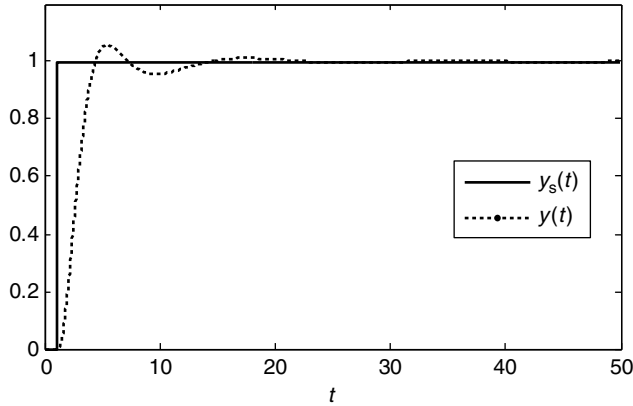


Figure 5.21 Control performance of a PID controller combined with an internal P controller tuned by the ITAE-2-setpoint and the OGM-unstable tuning methods based on the reduced models.

5.9 Consideration of Modeling Errors

If the process model has significant errors, then the PID controller should be tuned in a conservative way. The simplest method to incorporate the modeling error is to modify the model parameters in the direction that the tuning rules provide more conservative tuning parameters. From the tuning formula for the IMC, ITAE-1 and ITAE-2, it is clear that the tuning parameters become conservative as increasing k , τ and θ/τ of the FOPTD and SOPTD models. So, the recommendation is to use new adjustable parameters (e_k , e_τ and $e_{\theta/\tau}$) in the form $\bar{k} = k(1 + e_k)$, $\bar{\tau} = \tau(1 + e_\tau)$ and $\bar{\theta} = \theta(1 + e_\tau)(1 + e_{\theta/\tau})$. For example, if 5% modeling errors are assumed (that is, $e_k = e_\tau = e_{\theta/\tau} = 0.05$), then $\bar{k} = 1.05k$, $\bar{\tau} = 1.05\tau$ and $\bar{\theta} = 1.05^2\theta$ will be obtained. Then, the tuning rules on the basis of the modified model of $\bar{k} = 1.05k$, $\bar{\tau} = 1.05\tau$ and $\bar{\theta} = 1.05^2\theta$ would provide more conservative tuning parameters. For the ZN tuning rule, adjustable parameters (e_k and e_p) in the form of $\bar{k}_u = k_u/(1 + e_k)$ and $\bar{p}_u = p_u(1 + e_p)$ can be used to consider the modeling errors.

5.10 Concluding Remarks

Several simple PID tuning rules are introduced in this chapter. If the dynamics of the process are simple and a roughly tuned PID controller satisfies the control requirements, then trial-and-error tuning is sufficient. For a more systematic tuning, the ZN, IMC, ITAE-1 and ITAE-2 tuning rules are available for a stable process. The ZN tuning rule needs the ultimate frequency data of the process and the IMC and ITAE-1 tuning rules require the FOPTD model. The ITAE-2 needs the SOPTD model. Among the ZN, IMC and ITAE-1 tuning rules, the IMC tuning rule and the ITAE-1-disturbance show the best tuning result for the step setpoint change problem and for the step input disturbance rejection problem respectively. ITAE-2 shows almost the same responses as those of the optimal tuning. If the aim is for high performance or the process is underdamped, then the ITAE-2 tuning rule is recommended. Also, the optimal gain margin tuning rule for an unstable process is introduced. If the given process is high order, then it can be reduced to an FOPTD or SOPTD model using the model reduction method. Then, it is straightforward to tune

the PID controller using the above-mentioned tuning rules. The tuning strategy using model reduction is useful for the tuning of a PID controller combined with an internal feedback loop for cascade control, an integrating process and an unstable process.

Problems

5.1 Find the tuning parameters of a PID controller using the IMC and the ITAE-1 tuning rules for the process $G(s) = 1.5 \exp(-0.3s)/(10s + 1)$.

- (a) no modeling errors.
- (b) 3% modeling errors.

5.2 Reduce the process $G(s) = 2.0 \exp(-0.2s)/(s + 1)^3$ to an FOPTD model and tune the PID controller for the process using the IMC and the ITAE-1 tuning rules.

5.3 Find the tuning parameters of a PID controller using the ITAE-2 tuning rule for the process $G(s) = 5.0 \exp(-0.3s)/(2s^2 + 6.0s + 1.5)$.

5.4 Find the tuning parameters of a PID controller using the ITAE-2 tuning rule for the process $G(s) = 5.0(-0.2s + 1) \exp(-0.3s)/(2s^2 + 6.0s + 1.5)$.

5.5 Find the tuning parameters of a PID controller using the ZN tuning rule for the following processes:

(a) $2.0 \frac{dy(t)}{dt} + y(t) = 1.5u(t - 0.3)$

(b) $2.0 \frac{dy(t)}{dt} + y(t) = 1.5u(t - 0.3) + 5.0$

(c) $G(s) = \frac{3.0 \exp(-0.5s)}{2s^2 + 5s + 1}$

(d) $1.5 \frac{d^2y(t)}{dt^2} + 3.0 \frac{dy(t)}{dt} + 1.2y(t) = 1.0u(t - 0.1) + 0.5$

(e) $G(s) = \frac{2.0}{(s + 1)^5}$

(f) $G(s) = \frac{-2.0}{(s + 1)^3}$

5.6 Find the tuning parameters of a P and a PD controller for the process $G(s) = 2.0 \exp(-0.1s)/(3s - 1)(s + 1)$.

5.7 Reduce the process $G(s) = 2.0 \exp(-0.2s)/(s + 1)^3$ to an SOPTD model and tune the PID controller for the process using the ITAE-2 tuning rules. Simulate the control performance of the PID controller for a step setpoint change and a step input disturbance.

5.8 Find the tuning parameters of a PID controller using the ZN, IMC, ITAE-1 and ITAE-2 tuning rules and compare the control performances for the process $G(s) = 1.5 \exp(-0.3s)/(s + 1)(2s + 1)(3s + 1)$.

5.9 Find the tuning parameters of a PID controller using the ITAE-2 tuning rule and simulate the control performance for the step setpoint change at $t = 1.0$. You should define new deviation variables for the appropriate implementation of the PID controller. $y_0(t)$ is not measurable. The process output and the process input are $y(t)$ and $u(t)$ respectively.

$$\frac{d^2 y_0(t)}{dt^2} + 2 \frac{dy_0(t)}{dt} + y_0(t) = u(t - 0.3) + 1.0$$

$$y(t) = y_0(t) + 3.0$$

$$\left. \frac{dy_0(t)}{dt} \right|_{t=0} = 0, \quad \text{equivalently} \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = 0, \quad y(0) = 3.0, \quad u(t) = -1.0 \quad \text{for } t < 0$$

- 5.10 Run the virtual process of Process 1 (refer to the Appendix for details) and tune the PID controller using the trial-and-error tuning rule.
- 5.11 Run the virtual process of Process 1 and tune the PID controller using the continuous-cycling method.
- 5.12 Run the virtual process of Process 1 and tune the PID controller using the IMC tuning rule. Use the PRCmethod to estimate the FOPTD model.
- 5.13 Find the tuning parameters of a PID controller for the process of which the frequency responses are

- (a) $G(i0.0) = 1.5 - i0.0$, $G(i0.9) = 0.0 - i0.5$;
 (b) $G(i0.0) = 1.0 - i0.0$, $G(i0.3) = 0.307 - 0.668i$, $G(i0.6) = -0.122 - 0.391i$, $G(i0.9) = -0.158 - 0.175i$, $G(i1.2) = -0.126 - 0.0773i$.

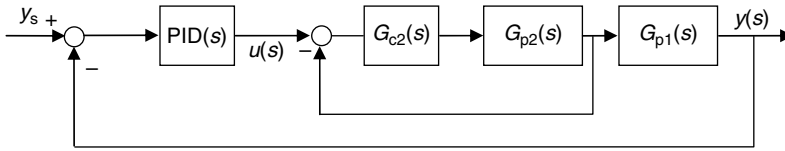


Figure P5.1

- 5.14 Tune the PID controller for the control system in Figure P5.1 and simulate the control performance for the step setpoint change. Here, $G_{c2}(s) = 4.0(1 + 0.3s)$, $G_{p2}(s) = \exp(-0.2s)/(s + 1)$, $G_{p1}(s) = \exp(-0.2s)/(s + 1)^3$.

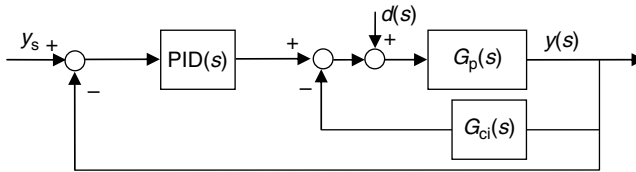


Figure P5.2

- 5.15 Tune the PID controller for the control system in Figure P5.2 and simulate the control performance for the step setpoint change and the step input disturbance.

- (a) $G_{ci}(s) = 6.36(1 + 0.224s)$, $G_p(s) = 2.0 \exp(-0.1s)/(3s - 1)(s + 1)$.
 (b) $G_p(s) = \exp(-0.2s)/s(s + 1)^3$, $G_{ci}(s) = 0.19$.

■

References

- Kwak, H.J., Sung, S.W. and Lee, I. (2000) Stabilizability conditions and controller design for unstable processes. *Chemical Engineering Research and Design*, **78**, 549.
- Lopez, A.M., Miller, C.L., Smith, C.L. and Murrill, P.W. (1967) Controller tuning relationships based on integral performance criteria, *Instrumentation Technology*, **14** (12), 72.
- Morari, M. and Zafiriou, E. (1989) *Robust Process Control*, Prentice-Hall, Englewood Cliffs, NJ.
- Sung, S.W. and Lee, I. (1996) Limitations and countermeasures of PID controllers. *Industrial & Engineering Chemistry Research*, **35**, 2596.
- Sung, S.W., Lee, O.J., Yu, S. and Lee, I. (1996) Automatic tuning of PID controller using second order plus time delay model. *Journal of Chemical Engineering of Japan*, **29**, 990.
- Ziegler, J.G. and Nichols, N.B. (1942) Optimum setting for automatic controllers, *Transactions of the American Society of Mechanical Engineers*, **64**, 759.

Bibliography

- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
- Stephanopoulos, G. (1984) *Chemical Process Control - An Introduction to Theory and Practice*, Prentice-Hall.

6

Dynamic Behavior of Closed-Loop Control Systems

It is important to predict the stability and the robustness to uncertainties in the case that a designed controller is applied to a process. This chapter defines the closed-loop transfer function and explains the relationship between the stability and the roots of the characteristic equation. Analysis tools for the Bode plot and the Nyquist plot are also introduced to predict the closed-loop stability by checking the open-loop transfer function. Also, the gain margin and the phase margin are defined to measure how much the closed-loop system is stable.

6.1 Closed-Loop Transfer Function and Characteristic Equation

A typical feedback control system has the structure shown in Figure 6.1. Here, $u(s)$ and $y(s)$ are the controller output and the process output respectively. $y_s(s)$ and $d(s)$ denote the setpoint and the disturbance respectively. $d_i(s)$ and $d_o(s)$ are called the input disturbance and the output disturbance respectively. $G_c(s)$ and $G(s)$ denote the transfer function of the controller and the process respectively. And $G_d(s)$ is the transfer function between the disturbance of $d(s)$ and the output disturbance of $d_o(s)$. The step setpoint change means that $y_s(s)$ is a step signal. The step input disturbance and the step output disturbance means $d_i(s)$ and $d_o(s)$ are step signals respectively.

6.1.1 Closed-Loop Transfer Function

In Figure 6.1, the transfer function from $y_s(s)$ to $y(s)$ is called the closed-loop transfer function between $y_s(s)$ and $y(s)$ because the loop is closed. Meanwhile, the transfer function from $e(s)$ to $y(s)$ is called the open-loop transfer function because the loop is open. Consider the following to derive the closed-loop transfer functions. The closed-loop transfer function from $y_s(s)$ to $y(s)$ is (6.1) when the signals of $d_i(s)$ and $d_o(s)$ are zero:

$$y(s) = G_c(s)G(s)e(s) = G_c(s)G(s)(y_s(s) - y(s)) \Rightarrow y(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)}y_s(s) \quad (6.1)$$

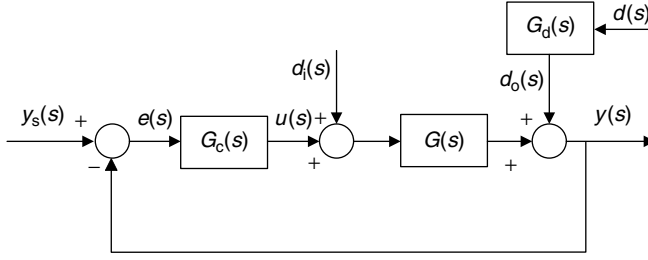


Figure 6.1 Typical feedback control system.

Equation (6.2) is the closed-loop transfer function from $d_i(s)$ to $y(s)$ when the signals of $y_s(s)$ and $d_o(s)$ are zero:

$$\begin{aligned} y(s) &= (G_c(s)e(s) + d_i(s))G(s) = G_c(s)G(s)(-y(s)) + d_i(s)G(s) \\ \Rightarrow y(s) &= \frac{G(s)}{1 + G_c(s)G(s)} d_i(s) \end{aligned} \quad (6.2)$$

Equation (6.3) is the closed-loop transfer function from $d(s)$ to $y(s)$ when the signals of $y_s(s)$ and $d_i(s)$ are zero:

$$y(s) = G_c(s)G(s)e(s) + d_o(s) = G_c(s)G(s)(-y(s)) + G_d(s)d(s) \Rightarrow y(s) = \frac{G_d(s)}{1 + G_c(s)G(s)} d(s) \quad (6.3)$$

By the superposition rule, the closed-loop transfer function from $y_s(s)$, $d_i(s)$, $d(s)$ to $y(s)$ is

$$y(s) = \frac{G_c(s)G(s)}{1 + G_c(s)G(s)} y_s(s) + \frac{G(s)}{1 + G_c(s)G(s)} d_i(s) + \frac{G_d(s)}{1 + G_c(s)G(s)} d(s) \quad (6.4)$$

6.1.2 Characteristic Equation

Note that all three closed-loop transfer functions have the same denominator of $1 + G_c(s)G(s)$. The characteristic equation is defined as $1 + G_c(s)G(s) = 0$. The roots of the characteristic equation correspond to the poles of the closed-loop transfer functions. So, the roots characterize the closed-loop dynamics and stability. If the real parts of all the roots are negative or a single root is located on the zero, then the closed-loop system is stable. If one of the real parts of the roots is positive or there is a multiple root (such as a double root, triple root, etc.) located on the zero, then the closed-loop system is unstable.

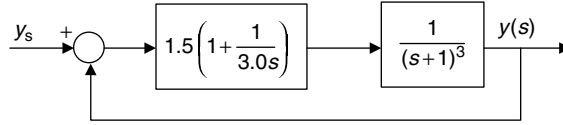


Figure 6.2 PI control system.

Example 6.1

Consider the control system in Figure 6.2.

The characteristic equation of Figure 6.2 is

$$1 + 1.5 \left(1 + \frac{1}{3.0s} \right) \frac{1}{(s+1)^3} = 0 \Rightarrow s(s+1)^3 + 1.5s + 1/3 = 0 \quad (6.5)$$

The roots are -2.103 , $-0.369 + 0.927i$, $-0.369 - 0.927i$ and -0.159 , which can be straightforwardly calculated by the roots function of the MATLAB function (i.e. roots ([1 3 2.5 1/3])). The real parts of all the roots are negative. So, the closed-loop system of Figure 6.2 is stable. Also, the two roots show nonzero imaginary parts. So, the closed-loop system will show an oscillatory response. For detailed descriptions on the relationship between the poles and the response of the process, refer to Chapter 3.

6.2 Bode Stability Criterion

Chapter 3 explains how to draw the Bode plot and Nyquist plot for the given process. In this section, the Bode plot and Nyquist plot are used to analyze the stability of the closed-loop control system. If the signals of the closed-loop control system diverge as time increases, then the system is called unstable. If all the signals converge, then it is called stable. The system is called marginally stable if the signals show continuous cycling.

Consider the typical control system in Figure 6.3. Here, the transfer function from $y_s(s)$ to $y(s)$ is called the closed-loop transfer function, denoted by $G_{CL}(s)$. Meanwhile, the transfer function from $e(s)$ to $y(s)$ is called the open-loop transfer function, denoted by $G_{OL}(s) = G_c(s)G(s)$.

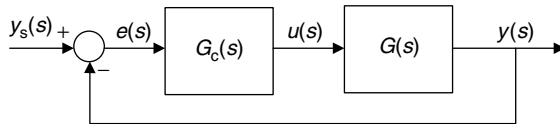


Figure 6.3 Block diagram of a typical closed-loop control system.

The Bode plot of the open-loop transfer function $G_{OL}(s) = G_c(s)G(s)$ can be used to analyze the stability of the closed-loop control system in Figure 6.3. Consider the following Bode stability criterion.

6.2.1 Bode Stability Criterion

$G_{OL}(s)$ is strictly proper and has no unstable poles. Also, $G_{OL}(s)$ has only a single critical frequency ω_c and a single gain crossover frequency ω_g . Then, the closed-loop control system in Figure 6.3 is stable if $|G_{OL}(i\omega_c)| < 1$. Otherwise, it is unstable. Here, the critical frequency

(phase crossover frequency) ω_c is defined as ω_c that satisfies $\angle G_{OL}(i\omega) = -\pi$. The gain crossover frequency ω_g is defined as ω that satisfies $|G_{OL}(i\omega)| = 1$.

A rigorous proof of the Bode stability criterion is omitted in this book. Instead, the Bode stability criterion can be understood conceptually by the following arguments. Note that the statements below are conceptually right. Strictly speaking, they are not complete from the mathematical point of view.

Let us start from Iteration 0 with the assumption of $y_s(t) = 0$. Iteration 0: assume that $e(t) = \sin(\omega_c t)$. Then, $y(t)$ becomes $y(t) = -|G_{OL}(i\omega_c)| \sin(\omega_c t)$ because $e(t) = \sin(\omega_c t)$ goes through the process dynamics that satisfy $\angle G_{OL}(i\omega_c) = -\pi$. Then, $e(t) = |G_{OL}(i\omega_c)| \sin(\omega_c t)$. Iteration 1: $y(t)$ becomes $y(t) = -|G_{OL}(i\omega_c)|^2 \sin(\omega_c t)$ because $e(t) = |G_{OL}(i\omega_c)| \sin(\omega_c t)$ goes through the process dynamics. Then, $e(t) = |G_{OL}(i\omega_c)|^2 \sin(\omega_c t)$. Iteration 2: $y(t)$ becomes $y(t) = -|G_{OL}(i\omega_c)|^3 \sin(\omega_c t)$ because $e(t) = |G_{OL}(i\omega_c)|^2 \sin(\omega_c t)$ goes through the process dynamics. Then, $e(t) = |G_{OL}(i\omega_c)|^3 \sin(\omega_c t)$. So, $y(t)$ exponentially diverges if $|G_{OL}(i\omega_c)| > 1$. $y(t)$ exponentially converges if $|G_{OL}(i\omega_c)| < 1$. If $|G_{OL}(i\omega_c)| = 1$, then the magnitude of $y(t)$ does not change (marginally stable).

Example 6.2

Consider the control system in Figure 6.4.

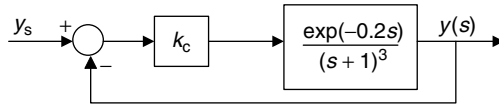


Figure 6.4 P control system.

The open-loop transfer function is $G_{OL}(s) = k_c \exp(-0.2s)/(s+1)^3$. Then, the following equations are obtained:

$$\angle G_{OL}(i\omega_c) = -0.2\omega_c - 3\tan^{-1}(\omega_c) = -\pi \Rightarrow \omega_c = 1.408 \quad (6.6)$$

$$|G_{OL}(i\omega_c)| = \frac{k_c}{(1+\omega_c^2)^{3/2}} \Rightarrow |G_{OL}(i\omega_c)| = \frac{k_c}{5.151} \quad (6.7)$$

The Bode plot of $G_{OL}(i\omega)$ with respect to k_c is shown in Figure 6.5. The scales of the y-axis and the x-axis are $\log_{10}(|G_{OL}(i\omega)|)$ and $\log_{10}(\omega)$ in the amplitude ratio plot. The scales of the y-axis and the x-axis are $\angle G_{OL}(i\omega)$ and $\log_{10}(\omega)$ in the phase-angle plot. For detailed descriptions on how to draw the Bode plot, refer to Chapter 3.

From (6.7) and Figure 6.5, it is concluded on the basis of the Bode stability criterion that the control system of Figure 6.4 is stable if $k_c < 5.151$, unstable if $k_c > 5.151$ and marginally stable if $k_c = 5.151$. Figure 6.6 shows the simulation results of the control system in Figure 6.4. It confirms that the Bode stability criterion is correct. As expected, the frequency of the oscillation (marginally stable) for $k_c = 5.151$ is $\omega_c = 1.408$ (equivalently, the period is $2\pi/\omega_c$).

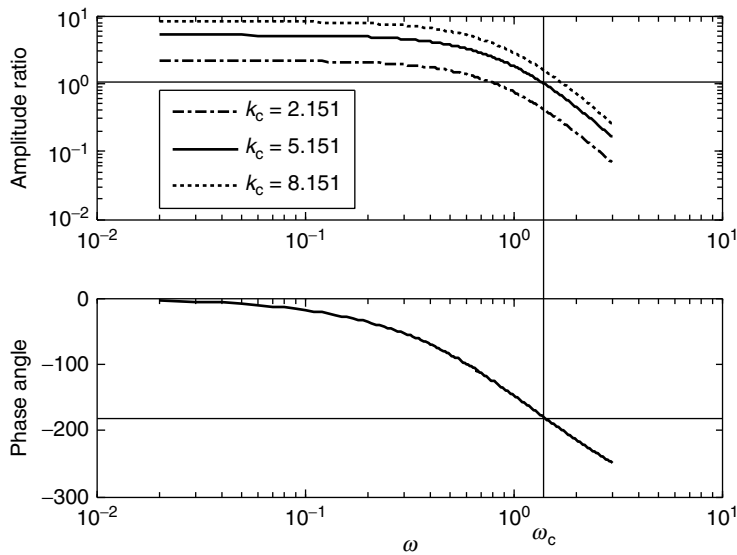


Figure 6.5 Bode plot of the P control system of Figure 6.4.

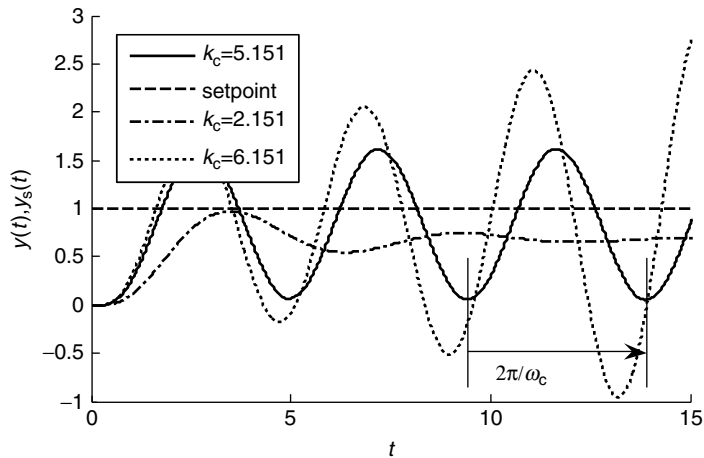


Figure 6.6 Simulation results of the P control system of Figure 6.4.

Example 6.3

Consider the PID control system in Figure 6.7.

The open-loop transfer function is $G_{OL}(s) = 2.5[1 + 1/(2.7s) + 0.675s] \exp(-0.1s)/(s + 1)^3$. The Bode plot of $G_{OL}(s)$ is shown in Figure 6.8.

The Bode plot shows that the control system is stable. Figure 6.9 shows the response of the control system of Figure 6.7 for the step setpoint change, which confirms that the Bode stability criterion is correct.

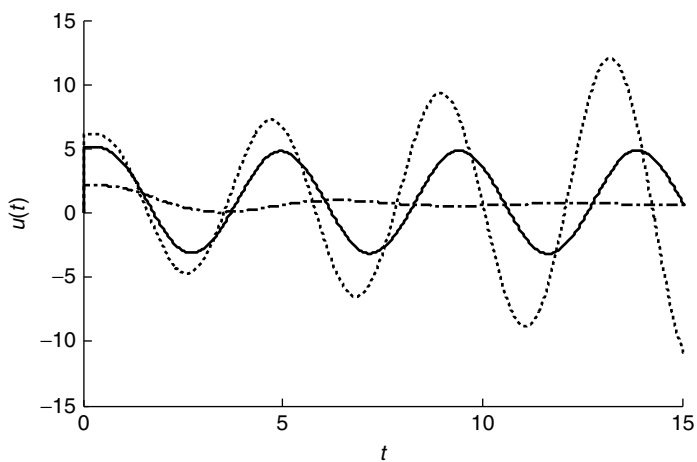


Figure 6.6 (Continued).

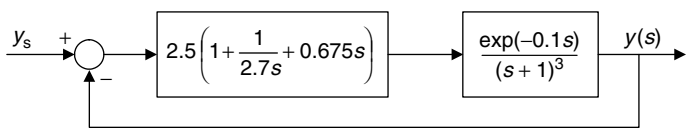


Figure 6.7 PID control system.

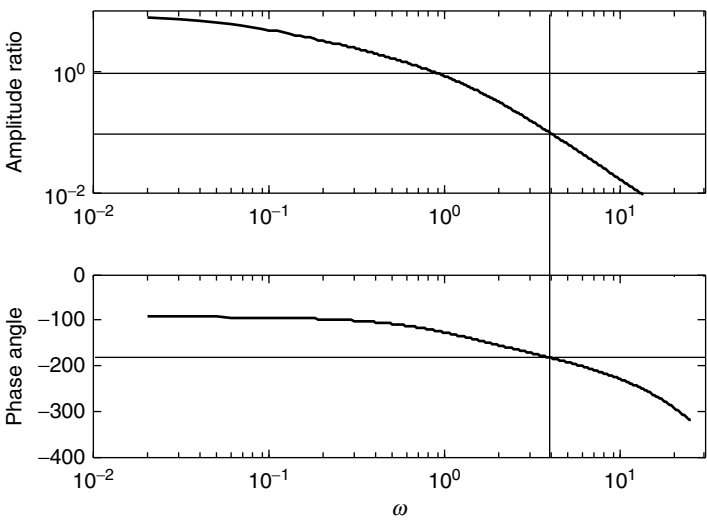


Figure 6.8 Bode plot of the PID control system of Figure 6.7.

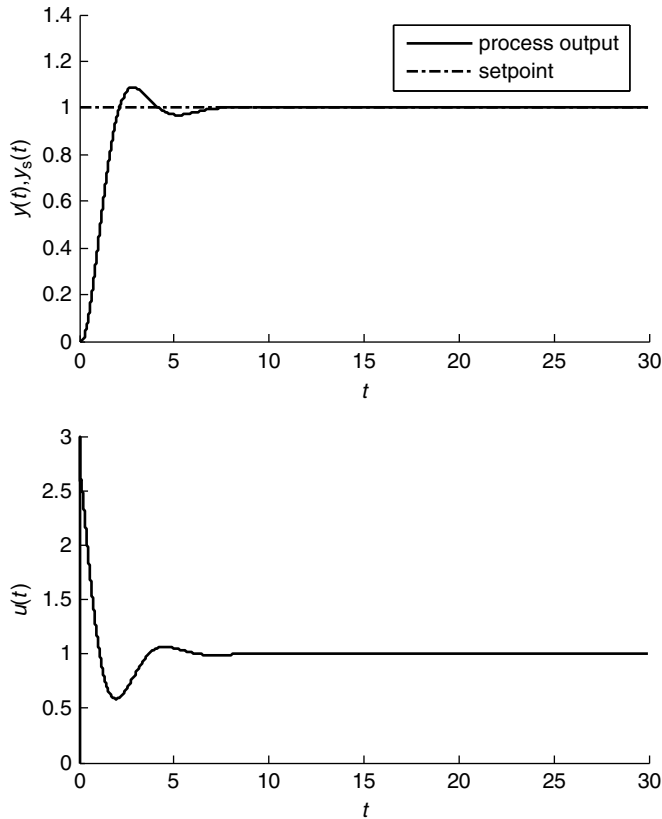


Figure 6.9 Simulation results of the PID control system of Figure 6.7.

6.3 Nyquist Stability Criterion

The Nyquist plot of the open-loop transfer function $G_{OL}(s) = G_c(s)G(s)$ can be used to analyze the stability of the closed-loop control system, as with the Bode plot. Consider the following Nyquist stability criterion.

6.3.1 Nyquist Stability Criterion

$G_{OL}(s)$ is strictly proper and has no unstable pole–zero cancellations. The Nyquist plot of $G_{OL}(s)$ encircles the $(-1, 0)$ point N times in the clockwise direction (N is negative for counterclockwise). Let P be the number of RHP poles of $G_{OL}(s)$. Then, $Z = N + P$ is the number of RHP roots of the characteristic equation. The closed-loop system is stable if and only if $Z = 0$.

Figure 6.10 shows the Nyquist plot for the control system in Figure 6.4. For detailed descriptions on how to draw the Nyquist plot, refer to Chapter 3.

$P = 0$ for the P control system of Figure 6.4 and the Nyquist plot encircles $(-1, 0)$ one time ($N = 1$) if $k_c > 5.151$. So, $Z = N + P = 1$ for $k_c > 5.151$, which means that the control system is

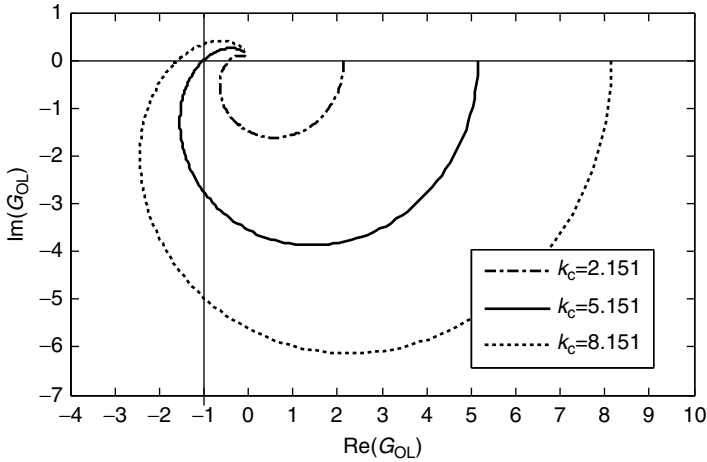


Figure 6.10 Nyquist plot for the control system of Figure 6.4.

unstable by the Nyquist stability criterion. $N=0$, $P=0$ and $Z=N+P=0$ for $k_c < 5.151$. So, the control system is stable for $k_c < 5.151$ by the Nyquist stability criterion.

Figure 6.11 shows the Nyquist plot for the control system in Figure 6.7.

$P=0$ for the PID control system of Figure 6.7 and $N=0$ as shown in Figure 6.11. So, the control system is stable by the Nyquist stability criterion. The same conclusion as that of the Bode stability criterion is obtained.

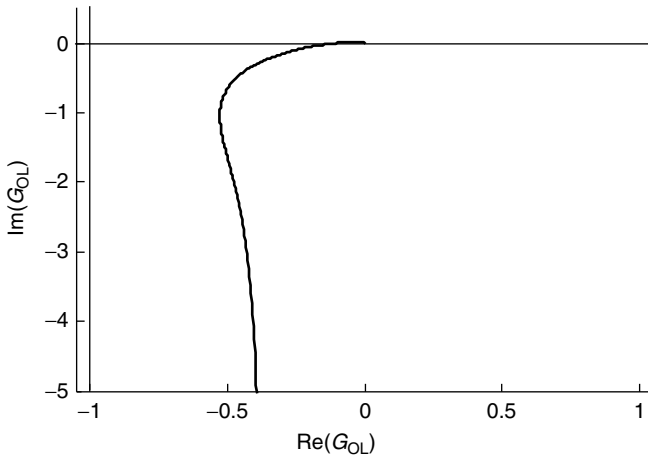


Figure 6.11 Nyquist plot for the control system of Figure 6.7.

Example 6.4

Consider the control system in Figure 6.12.

The open-loop transfer function is $G_{OL}(s) = 3 \exp(-0.3s)/(3s - 1)$. The MATLAB code to plot the Nyquist plot of the open-loop transfer function, the Nyquist plot and the simulation

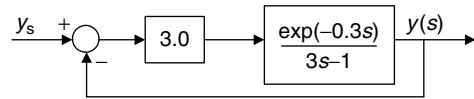


Figure 6.12 P control system.

result of the closed-loop control system of Figure 6.12 are shown in Table 6.1, Figures 6.13 and Figure 6.14 respectively. Note that $G_{OL}(s)$ has one RHP pole and the Nyquist plot starts at the $(-3, 0)$ point and encircles the $(-1, 0)$ point in the counterclockwise direction. That is, $P = 1$ and $N = -1$. Then, on the basis of $Z = N + P = 0$, it can be concluded that the control

Table 6.1 MATLAB code to plot the Nyquist plot of Example 6.4.

<pre>nyquist_ex1.m clear; w_max=30.0; delw=0.01; n=round(w_max/delw); w=0; G=g_nyquist_ex1(w); m=1; W(m)=w; R(m)=real(G); I(m)=imag(G); for m=2:n w=m*delw; G=g_nyquist_ex1(w); W(m)=m*delw; R(m)=real(G); I(m)=imag(G); end figure(1); plot(R,I);</pre>	<pre>g_nyquist_ex1.m function [g]=g_nyquist_ex1(w) s=i*w; g=3.0*exp(-0.3*s)/(3*s-1); end command window >> nyquist_ex1</pre>
--	---

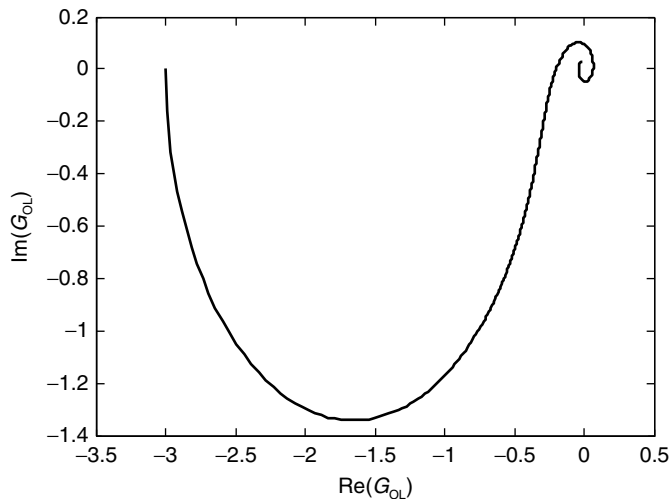


Figure 6.13 Nyquist plot for the control system of Figure 6.12.

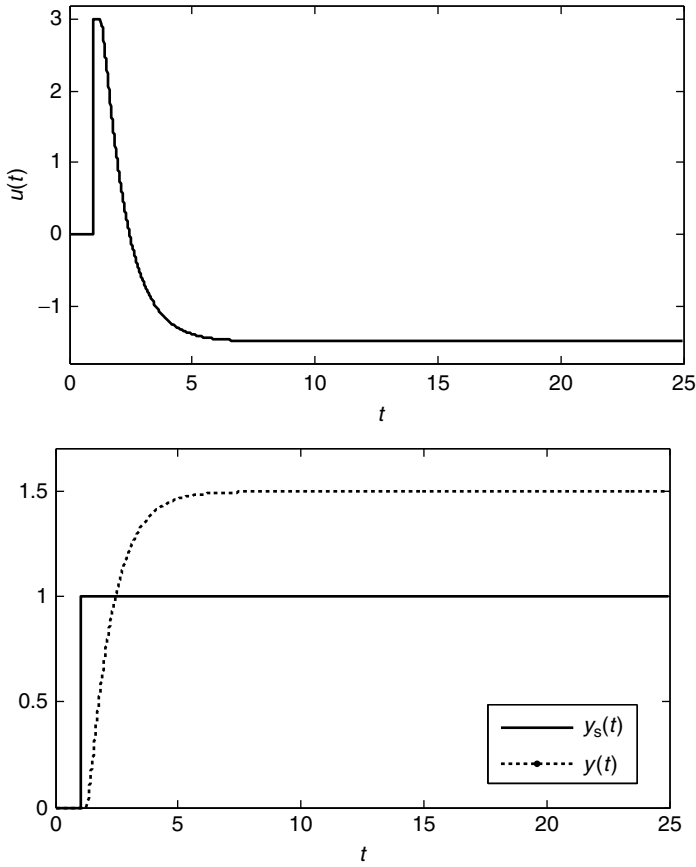


Figure 6.14 Simulation results of the P control system of Figure 6.12.

system is stable. The simulation result of the control system in Figure 6.14 confirms that the Nyquist stability criterion is correct. In this case, the Bode stability criterion cannot be applied because the open-loop process has an unstable pole.

6.4 Gain Margin and Phase Margin

The gain margin (GM) and the phase margin (PM) are quantitative measures to indicate how much the control system is stable. GM is defined as the reciprocal of the amplitude ratio of the open-loop transfer function at the critical frequency. That is, $GM = 1/|G_{OL}(i\omega_c)|$. PM is defined as the phase difference between $-\pi$ and the phase angle of the open-loop transfer function at the gain crossover frequency. That is, $PM = \angle G_{OL}(i\omega_g) + \pi$. The control system is more stable as GM and PM increase.

Figures 6.15 and 6.16 show GM and PM in the Bode plot and the Nyquist plot.

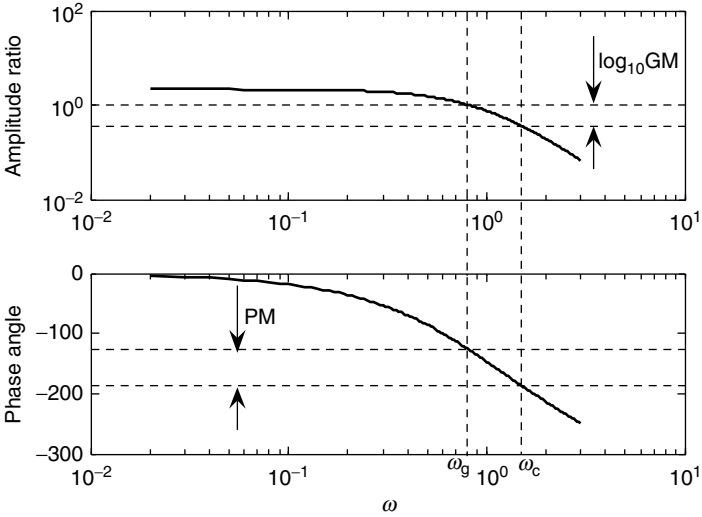


Figure 6.15 GM and PM in the Bode plot.

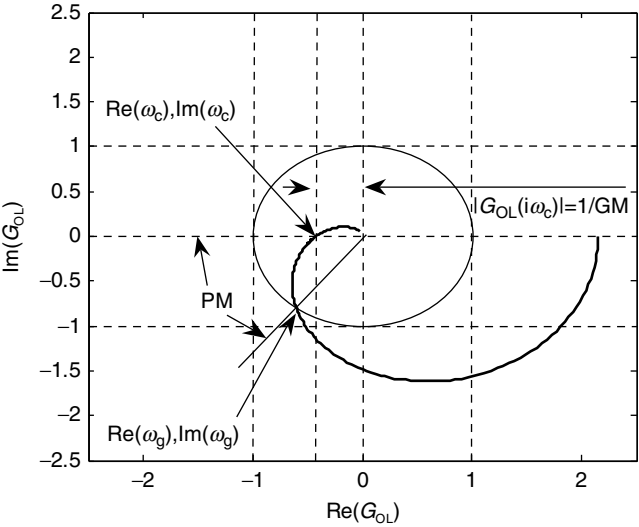


Figure 6.16 GM and PM in the Nyquist plot.

Example 6.5

Estimate the GM and the PM for the control system in Figure 6.4 with $k_c = 2.151$.

Solution The open-loop transfer function is $G_{OL}(s) = 2.151 \exp(-0.2s)/(s + 1)^3$. Then, the critical frequency is obtained by solving

$$\angle G_{OL}(i\omega_c) = -0.2\omega_c - 3\tan^{-1}(\omega_c) = -\pi \text{ or } \text{Im}(G_{OL}(i\omega_c)) = 0.0 \Rightarrow \omega_c = 1.408 \quad (6.8)$$

And, the gain crossover frequency is obtained by solving

$$|G_{OL}(i\omega_g)| = \frac{2.151}{(1 + \omega_g^2)^{3/2}} = 1 \Rightarrow \omega_g = 0.8164 \quad (6.9)$$

Because $|G_{OL}(i\omega_c)| = 0.4176$, $GM = 1/|G_{OL}(i\omega_c)| = 2.3945$. $\angle G_{OL}(i\omega_g) = \arctan 2(-0.7981, -0.6023) = -2.2173$ is obtained from $G_{OL}(i\omega_g) = -0.6023 - i0.7981$. So, $PM = -2.2173 + \pi = 0.9243 = 52.96^\circ$.

Example 6.6

Estimate the GM and the PM for the control system in Figure 6.7.

Solution The open-loop transfer function is $G_{OL}(s) = 2.5[1 + 1/(2.7s) + 0.675s] \exp(-0.1s)/(s + 1)^3$. Then, the critical frequency is obtained by solving

$$\text{Im}(G_{OL}(i\omega_c)) = 0.0 \Rightarrow \omega_c = 3.836 \quad (6.10)$$

And the gain crossover frequency is obtained by solving

$$|G_{OL}(i\omega_g)| = 1 \Rightarrow \omega_g = 0.936 \quad (6.11)$$

Because $|G_{OL}(i\omega_c)| = 0.1078$, $GM = 1/|G_{OL}(i\omega_c)| = 9.278$. $\angle G_{OL}(i\omega_g) = \arctan 2(-0.8532, -0.5208) = -2.1188$ is obtained from $G_{OL}(i\omega_g) = -0.5208 - i0.8532$. So, $PM = -2.1188 + \pi = 1.0228 = 58.60^\circ$.

Problems

6.1 Obtain the characteristic equation for the control system in Figure P6.1 and determine if it is stable or unstable. Also, explain the effects of the parameters of the PID controller on the stability of the closed-loop system. Here, the process is $G_p(s) = 1/(s + 1)^4$.

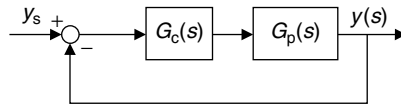


Figure P6.1

- (a) $G_c(s) = 1.5 \left(1 + \frac{1}{3.0s} \right)$
- (b) $G_c(s) = 3.0 \left(1 + \frac{1}{3.0s} \right)$
- (c) $G_c(s) = 3.0 \left(1 + \frac{1}{10.0s} \right)$
- (d) $G_c(s) = 3.0 \left(1 + \frac{1}{3.0s} + 0.5s \right)$.

6.2 Determine if the closed-loop system of Figure P6.1 is stable or unstable using the Bode and Nyquist stability criterion. If it is stable, find the GM and PM.

- (a) $G_p(s) = \frac{\exp(-0.5s)}{(s+1)^2}$, $G_c(s) = 1.5$
- (b) $G_p(s) = \frac{\exp(-0.5s)(1+0.2s)}{(s+1)^3}$, $G_c(s) = 1.5\left(1 + \frac{1}{3.0s}\right)$
- (c) $G_p(s) = \frac{\exp(-0.5s)}{(s+1)^2}$, $G_c(s) = 10.0$
- (d) $G_p(s) = \frac{\exp(-0.5s)}{(s+1)^2}$, $G_c(s) = 1.5(1+0.5s)$
- (e) $G_p(s) = \frac{\exp(-0.5s)(1+0.5s)}{(s+1)^3}$, $G_c(s) = 2.0\left(1 + \frac{1}{3.0s} + 0.7s\right)$
- (f) $G_p(s) = \frac{\exp(-0.2s)}{s(s+1)^3}$, $G_c(s) = 0.19$

6.3 Determine if the closed-loop system of Figure P6.1 is stable or unstable.

- (a) $G_p(s) = \frac{2.0 \exp(-0.1s)}{(3s-1)(s+1)}$, $G_c(s) = 6.36(1+0.224s)$
- (b) $G_p(s) = \frac{\exp(-0.2s)}{(s+1)(2s-1)}$, $G_c(s) = 1.5$

6.4 Simulate Problem 6.2 and confirm if the closed-loop response coincides with your expectation.

6.5 Simulate Problem 6.3 and confirm if the closed-loop response coincides with your expectation.

References

- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
 Stephanopoulos, G. (1984) *Chemical Process Control - An Introduction to Theory and Practice*, Prentice-Hall.

Enhanced Control Strategies

7.1 Cascade Control

Cascade control uses an additional internal feedback loop to reject disturbances more effectively. The typical structure of a cascade control is shown in Figure 7.1a. It is composed of a primary (master) controller $G_{c1}(s)$ and a secondary (slave) controller $G_{c2}(s)$. $G_{p1}(s)$ and $G_{p2}(s)$ are the primary process and the secondary process respectively. The output $y_2(s)$ of the secondary process $G_{p2}(s)$ is measurable. $u_1(s)$ and $u_2(s)$ are the control outputs of the primary controller and the secondary controller respectively. $y_{s1}(s)$ and $y_{s2}(s)$ are the setpoints of the primary controller and the secondary controller respectively. Note that the control output of the primary controller is the setpoint of the secondary controller; that is, $u_1(s) = y_{s2}(s)$.

In Figure 7.1a, the disturbance $d(t)$ directly affects $y_2(t)$. Meanwhile, $d(t)$ indirectly affects $y_1(t)$ because it goes through the dynamics of $G_{p1}(s)$. The cascade control system in Figure 7.1a can detect the disturbance more quickly by measuring $y_2(t)$, compared with the conventional control system in Figure 7.1b. If the dynamics of $G_{p2}(s)$ are fast, then the internal feedback loop (the secondary controller) can reject the disturbance quickly before the disturbance affects $y_1(t)$, resulting in an improved disturbance rejection performance. Meanwhile, the conventional control system inevitably shows a slow response to the disturbance because it measures only $y_1(t)$. The following three conditions should be satisfied for a successful cascade control:

1. The output of the secondary process $y_2(s)$ is measurable.
2. The disturbance affects the output of the secondary process $y_2(s)$ more quickly than the output of the primary process $y_1(s)$.
3. The dynamics of the secondary process are fast enough for the secondary control action to remove the effect of the disturbance quickly.

Figure 7.2 compares a conventional PID controller and cascade control, wherein $G_{p1}(s) = \exp(-0.1s)/(s + 1)^3$, $G_{p2}(s) = \exp(-0.1s)/(0.5s + 1)$, $G_{c1}(s) = 2.5(1 + 1/2.7s + 0.675s)$, $G_{c2}(s) = 5.0$ and $d(s) = 1/s$. As shown in Figure 7.2, the cascade control rejects the disturbance more quickly than the conventional control because $u_2(t)$ of the cascade control quickly attenuates the disturbance.

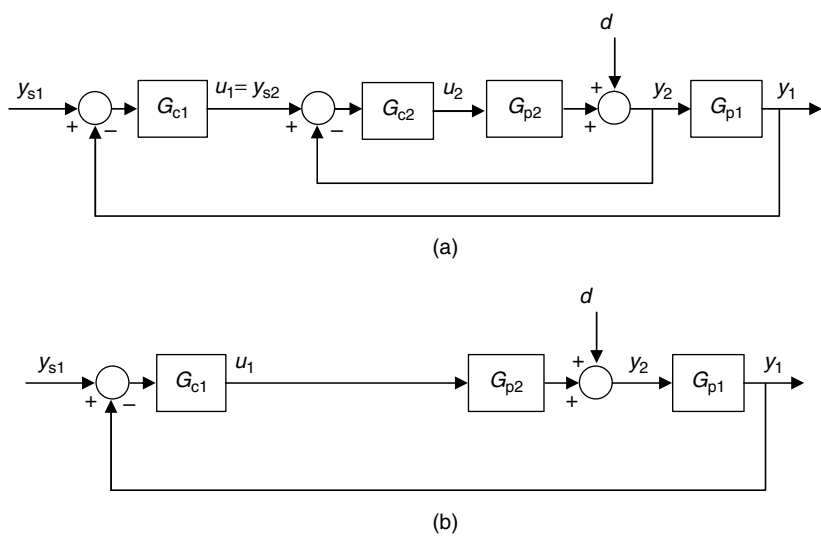


Figure 7.1 (a) Cascade control system and (b) conventional control system.

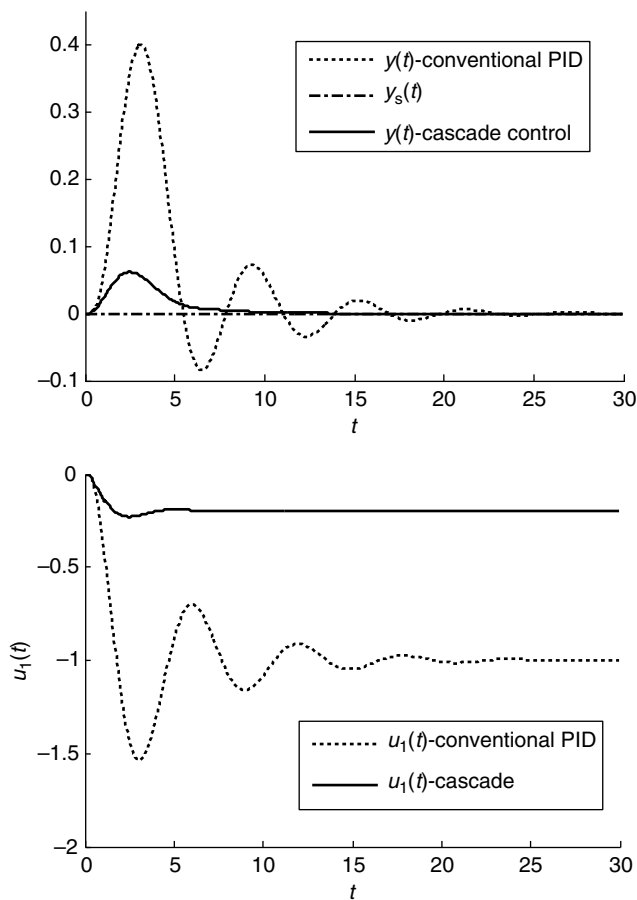


Figure 7.2 Control performances of a cascade control and a conventional PID control.

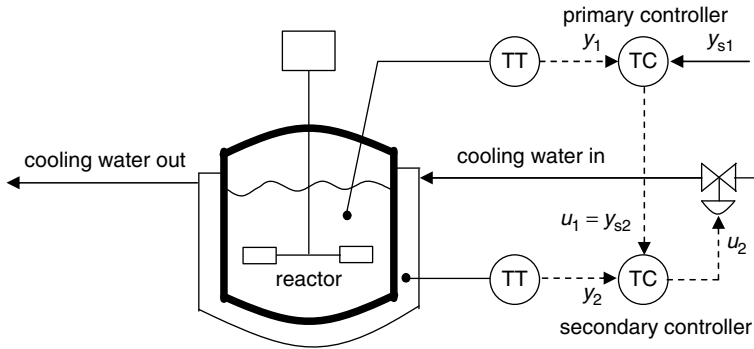


Figure 7.3 Cascade control for an exothermic batch reactor.

Figure 7.3 shows a typical cascade control for an exothermic batch reactor, where TT and TC represent the temperature transmitter and the temperature controller respectively. The input and the output of the primary process are $y_2(t)$ and $y_1(t)$ respectively. $u_2(t)$ and $y_2(t)$ are the input and the output of the secondary process respectively.

In Figure 7.3, the disturbance is the temperature variation of the cooling water. This directly affects the temperature $y_2(t)$ of the cooling water in the jacket. $u_2(t)$ affects $y_2(t)$ quickly. So, the secondary controller can reject the disturbance quickly before the disturbance affects $y_1(t)$, resulting in excellent disturbance rejection performances.

Example 7.1

Simulate the cascade control system of Figure 7.2.

Solution The MATLAB code to simulate Figure 7.2 is shown in Table 7.1.

7.2 Time-Delay Compensators

If the time delay of the process is long, then there is no choice but to wait as long as the time delay to detect the effects of the present control action on the process output. Then, an aggressive control action is not possible because there is no chance to correct the side effects of the aggressive present action for the long time. So, the time delay is one of the most serious bottlenecks in improving control performance. Fortunately, if a model is available, then the effects of the present control action on the future process output can be predicted without waiting as long as the time delay by solving the differential equation of the model using a computer. In this section, the two approaches of the Smith predictor (Smith, 1957) and the decoupled predictor (Sung and Lee, 1996) for the time-delay compensation are introduced.

7.2.1 Smith Predictor

Consider the control system shown in Figure 7.4, where $G^*(s)$ is the time-delay-free process and $G_m^*(s)$ is the time-delay-free model. $G(s) = \exp(-\theta s)G^*(s)$ is the process. $G_c(s)$ is usually

Table 7.1 MATLAB code to simulate the cascade control system of Figure 7.2.

<pre> cascade_ex1.m clear; tf=30; delt=0.05; uu1=zeros(1,500); uu2=zeros (1,500); yy2=zeros(1,500); x1=zeros(3,1); x2=zeros(1,1); tf_k=round(tf/delt); y1=0.0; y1b=0.0; s1=0.0; ys1=0.0; ys1b=0.0; y2=0.0; y2b=0.0; kc1=2.5; ti1=2.7; td1=ti1/4; %primary PID kc2=5.0; %secondary P for k=1:tf_k t=(k-1)*delt; T(k)=t; Y1(k)=y1; Y2(k)=y2; Ys1(k)=ys1; U1(k)=uu1(500); U2(k)=uu2(500); for i=1:499 uu1(i)=uu1(i+1); end for i=1:499 uu2(i)=uu2(i+1); end for i=1:499 yy2(i)=yy2(i+1); end s1=s1+(kc1/ti1)*(ys1-y1)*delt; uu1(500)=kc1*(ys1-y1)+s1 +kc1*td1*(ys1-y1-ys1b+y1b)/delt; uu2(500)=kc2*(uu1(500)-y2); yy2(500)=y2; y1b=y1; ys1b=ys1; [x2,y2]=cc_process2(x2,delt, uu2); %cascade %[x2,y2]=cc_process2(x2,delt, uu1); %conventional y2=y2+1.0; %disturbance [x1,y1]=cc_process1(x1,delt, yy2); end figure(1); hold on; plot(T,Y1, T,Ys1); figure(2); hold on; plot(T,U1); </pre>	<pre> cc_process1.m function [next_x,y]=cc_process1 (x,delt,u); subdelt=delt/5; n=round (delt/subdelt); A=[0 0 -1; 1 0 -3; 0 1 -3]; B=[1; 0; 0]; C=[0 0 1]; delay=0.1; delay_k=round(delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return </pre>
	<pre> cc_process2.m function [next_x,y]=cc_process2 (x,delt,u); subdelt=delt/5; n=round (delt/subdelt); A=[-2]; B=[2]; C=[1]; delay=0.1; delay_k=round(delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return </pre>
	<pre> command window >> cascade_ex1 </pre>

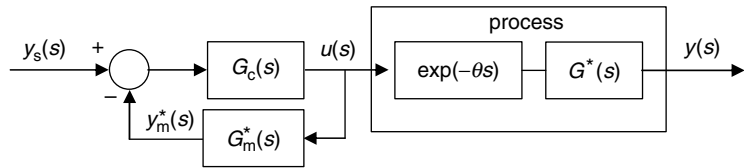


Figure 7.4 Predictive control using a process model.

$G_m^*(s) + G_p(s) - G_m(s)$ in (7.1) to the equivalent gain plus time delay like $G_m^*(s) + G_p(s) - G_m(s) \approx G_m^*(s)k_{eq}\exp(-\theta_{eq}s)$ to consider the modeling error. Then, reasonable controller gains by applying the usual PID tuning rules to $k_{eq}G_m^*(s)\exp(-\theta_{eq}s)$, $k_{eq} \geq 1.0$ and $\theta_{eq} \geq 0.0$. k_{eq} and θ_{eq} are adjustable parameters for the tuning of the Smith predictor. A more conservative controller will be obtained as the equivalent gain and the equivalent time delay are increased. In the implementation step, it is recommended to use $k_{eq}G_m^*(s)$ and $k_{eq}G_m^*(s)\exp(-(\theta_m + \theta_{eq})s)$ for the time-delay-free model and the process model of the Smith predictor in Figure 7.6. The recommendation is for a conservative operation of the Smith predictor.

The Smith predictor shows a good setpoint tracking performance because $G_c(s)$ is strongly tuned and the time-delay-free model output is used. Note that (7.1) becomes $1 + G_c(s)G_m^*(s) = 0$ if there are no modeling errors. So, it is clear that $G_c(s)$ can be strongly tuned because $G_m^*(s)$ has no time delay. But, it should be noted that the modeling error of $G_p(s) - G_m(s)$ is amplified by the high-gain controller of $G_c(s)$, as shown in (7.1). As a result, the closed-loop stability tends to be very sensitive to the modeling error $G_p(s) - G_m(s)$. Then, a small modeling error can make the closed-loop system unstable.

In summary, the Smith predictor can provide an excellent setpoint tracking performance if the process model is accurate. But, a small model error can destabilize the closed-loop system if the gains of the controller are tuned too strongly. The Smith predictor can be tuned in a reasonable way by considering the modeling error in the form of the equivalent gain plus time delay.

Example 7.2

Simulate the Smith predictor of Figure 7.6 with $G(s) = \exp(-1.8s)/(s^2 + 2s + 1)$, $G_m(s) = 0.95 \exp(-1.7s)/(s^2 + 2.2s + 1)$. Assume 7% modeling error for the equivalent gain and equivalent time delay; that is, $G_m^*(s) + G_p(s) - G_m(s) \approx G_m^*(s)1.07\exp(-1.7 \times 0.07s)$. Compare the Smith predictor with the conventional PID controller.

Solution $G_m^*(s)1.07\exp(-1.7 \times 0.07s) = 0.95 \times 1.07\exp(-1.7 \times 0.07s)/(s^2 + 2.2s + 1)$ should be used to tune the PID controller of the Smith predictor. Then, the ITAE-2 tuning rule provides $G_c(s) = 8.455(1 + 1/2.270s + 0.456s)$, and $G_m(s) = 0.95 \times 1.07\exp(-1.7 \times 1.07s)/(s^2 + 2.2s + 1)$ and $G_m^*(s) = 0.95 \times 1.07\exp(-0.0s)/(s^2 + 2.2s + 1)$ should be used for the models of the Smith predictor. The conventional PID controller designed by the ITAE-2 tuning rule based on the model $G_m(s) = 0.95 \exp(-1.7s)/(s^2 + 2.2s + 1)$ is $G_c(s) = 1.027(1 + 1/2.560s + 0.751s)$. The MATLAB code for simulation and the simulation results are shown in Table 7.2 and Figure 7.7 respectively. Note that the closed-loop response is very fast because the parameters of the PID controller of the Smith predictor can be tuned strongly, whereas the conventional PID controller shows a slow closed-loop response, as shown in Figure 7.6.

7.2.2 Time-Delay Compensator with Decoupled Control Structure

The high-gain controller of the Smith predictor amplifies the modeling error, possibly, resulting in unstable closed-loop responses for small errors. To overcome the problem, the amplification phenomenon by the high-gain controller should be removed. The time delay compensation with decoupled control structure can be a good candidate to solve the problem (Sung and Lee, 1996).

Table 7.2 MATLAB code to simulate the Smith predictor.

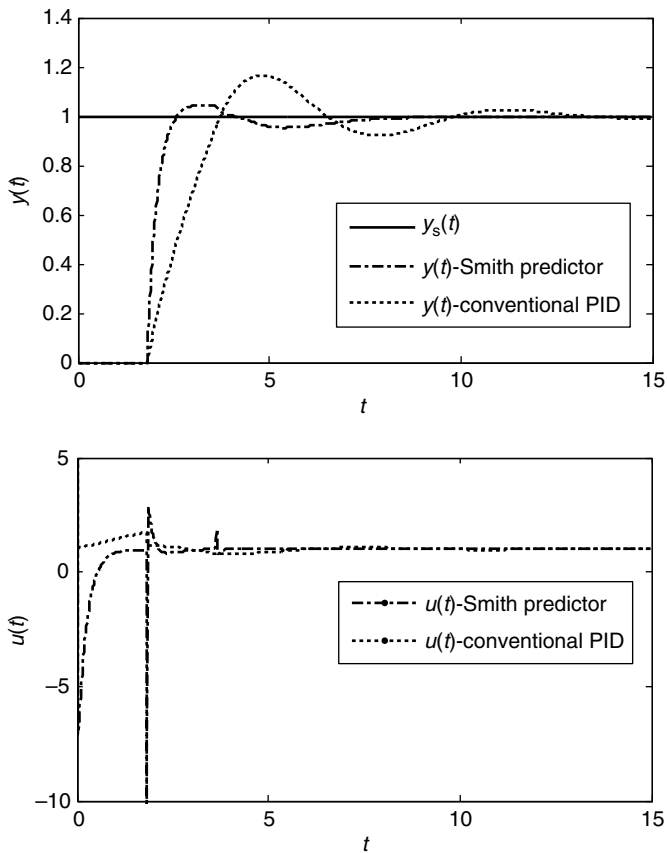
<pre>smith_ex1.m clear; tf=15; delt=0.01; u=zeros(1,500); x=zeros(2,1); xm=zeros(2,1); xm_free=zeros(2,1); tf_k=round(tf/delt+0.0000001); s=0.0; ys=1.0; ysb=0.0; y=0.0; yb=0.0; ym=0.0; ymb=0.0; ym_free=0.0; ymb_free=0.0; kc=8.455; ti= 2.270; td= 0.456; %Smith %kc=1.027; ti= 2.560; td= 0.751; %Conventional PID for k=1:tf_k t=(k-1)*delt; T(k)=t; Y(k)=y; Ys(k)=ys; U(k)=u(500); for i=1:499 u(i)=u(i+1); end e_pid=ys-y+ym-ym_free; eb_pid=ysb-yb+ymb-ymb_free; s=s+(kc/ti)*e_pid*delt; u(500)=kc*e_pid+s+kc*td*(e_pid-eb_pid)/delt; yb=y; ymb=ym; ymb_free=ym_free; ysb=ys; % Remove the below two lines for the conventional PID [xm,ym]=smith_model_ex1(xm,delt,u); % model [xm_free,ym_free]=smith_model_delay_free_ex1(xm_free,delt,u); [x,y]=smith_process_ex1(x,delt,u); % process end figure(1); plot(T,Ys,T,Y); hold on; %figure(1); plot(T,Y); hold on; figure(2); plot(T,U); hold on; return</pre>	
<pre>smith_process_ex1.m function [next_x,y]=smith_process_ ex1(x,delt,u); subdelt=delt/5; n=round(delt/ subdelt); A=[0 -1; 1 -2.0]; B=[1.00; 0]; C=[0 1]; delay=1.80; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return</pre>	<pre>smith_model_ex1.m function [next_x,y]=smith_model_ ex1(x,delt,u); subdelt=delt/5; n=round(delt/ subdelt); A=[0 -1; 1 -2.2]; B=[0.95*1.07; 0]; C=[0 1]; delay=1.7*1.07; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return</pre>
<pre>smith_model_delay_free.m function</pre>	<pre>command window >> smith_ex1</pre>

Table 7.2 (Continued)

```

[next_x,y]=smith_model_
delay_free_ex1(x,delt,u);
subdelt=delt/5; n=round(delt/
subdelt);
A=[0 -1; 1 -2.2]; B=[0.95*1.07; 0];
C=[0 1];
for i=1:n
    dx=A*x+B*u(500);
    x=x+dx*subdelt;
end
next_x=x; y=C*x;
return

```

**Figure 7.7** Control performances of the Smith predictor and the conventional PID controller in Example 7.2.

Consider the control structure shown in Figure 7.8. This is composed of two PID controllers, a time-delay-free model and a process model, where $y(s)$, $y_m(s)$ and $y_m^*(s)$ denote the process output, the model output and the time-delay-free model output respectively. $d(s)$ is the input disturbance. The left-hand side is for the setpoint change. $y_m(s)$ corresponds to the process output by the setpoint change and then $y(s) - y_m(s)$ corresponds to the process output by the disturbance. So, $G_{c,s}(s)$, for which the setpoint is $y_s(s)$, controls the time-delay-free model output of $y_m^*(s)$. $G_{c,d}(s)$, for which the setpoint is zero, rejects the disturbance and the modeling error.

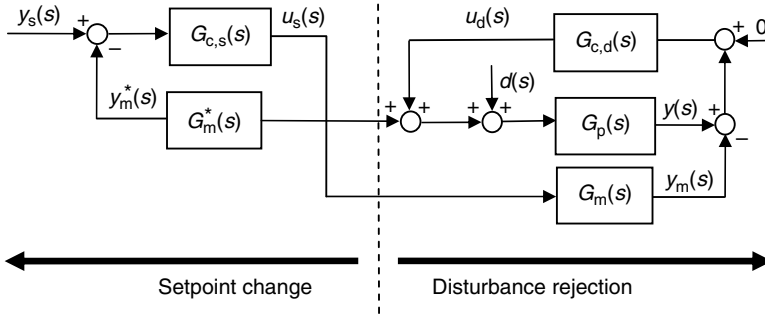


Figure 7.8 Time-delay compensation using a decoupled control structure.

Consider the two closed-loops in Figure 7.8. Clearly, the two closed-loops are decoupled from the stability point of view. The first closed-loop for the setpoint change includes only $G_{c,s}(s)$ and $G_m^*(s)$ and the characteristic equation is $1 + G_{c,s}(s)G_m^*(s) = 0$. So, the controller $G_{c,s}(s)$ should be tuned on the basis of $G_m^*(s)$. Then, the tuning parameters would be strongly tuned because $G_m^*(s)$ is the time-delay-free model. Meanwhile, the second closed-loop for the disturbance rejection includes only $G_{c,d}(s)$ and $G_p(s)$ and the characteristic equation is $1 + G_{c,d}(s)G_p(s) = 0$. So, the controller $G_{c,d}(s)$ should be tuned on the basis of the process model $G_m(s)$. Then, the controller $G_{c,d}(s)$ would be tuned in a conservative way because $G_m(s)$ includes the time delay. Now, it is clear that the amplification phenomenon of the modeling error ($G_p(s) - G_m(s)$) by the high-gain controller in the Smith predictor is completely removed by the control structure of Figure 7.8 because it manipulates the disturbance rejection using a low-gain controller $G_{c,d}(s)$. It is concluded that the time-delay compensator of Figure 7.8 is superior to the Smith predictor of Figure 7.6.

Example 7.3

Simulate the Smith predictor and the decoupled time-delay compensator of Figure 7.8 with $G(s) = \exp(-1.95s)/(s^2 + 2s + 1)$, $G_m(s) = 0.95 \exp(-1.7s)/(s^2 + 2.2s + 1)$. Assume 3% modeling error for the equivalent gain and equivalent time delay; that is, $G_m^*(s) + G_p(s) - G_m(s) \approx G_m^*(s)1.03 \exp(-1.7 \times 0.03s)$.

Solution $G_m^*(s)1.03 \exp(-1.7 \times 0.03s) = 0.95 \times 1.03 \exp(-1.7 \times 0.03s)/(s^2 + 2.2s + 1)$ is used to tune $G_{c,s}(s)$ in Figure 7.8 and $G_c(s)$ in Figure 7.6. Then, the ITAE-2-setpoint tuning rule provides $G_c(s) = 18.29(1 + 1/2.265s + 0.436s)$ for Figure 7.6 and $G_{c,s}(s) = 18.29$

$(1 + 1/2.265s + 0.436s)$ for Figure 7.8, and $G_m(s) = 0.95 \times 1.03 \exp(-1.7 \times 1.03s)/(s^2 + 2.2s + 1)$ and $G_m^*(s) = 0.95 \times 1.03 \exp(-0.0s)/(s^2 + 2.2s + 1)$ are used for the models in Figures 7.6 and 7.8. $G_{c,d}(s)$ designed by the ITAE-2-disturbance tuning rule based on the model $G_m(s) = 0.95 \times 1.0 \exp(-1.7 \times 1.03s)/(s^2 + 2.2s + 1)$ is $G_c(s) = 1.261(1 + 1/2.370s + 0.878s)$. The MATLAB code for simulation is shown in Table 7.3. Figure 7.9 confirms a poor robustness of the Smith predictor to the modeling error. The decoupled time-delay compensator of Figure 7.8 shows acceptable robustness because the setpoint tracking problem and the disturbance (modeling error) rejection problem are decoupled.

Table 7.3 MATLAB code to simulate the decoupled time-delay compensator in Example 7.3.

```

                                delay_compensator_ex1.m
clear;
tf=20; delt=0.005; tf_k=round(tf/delt+0.0000001);
cont_out1=zeros(1,500); %control ouput1
cont_out2=zeros(1,500); %control ouput2
x1=zeros(2,1); x2=zeros(2,1); x3=zeros(2,1);
y=0.0; yb=0.0; s1=0.0; yset=1.0; ysb=0.0; ys=0.0;
ym=0.0; ymb=0.0; ym_star=0.0; ymb_star=0.0; s2=0.0;
kc1=18.285; ti1= 2.265; td1= 0.436;
kc2=1.261; ti2=2.370; td2=0.878;
for k=1:tf_k
    t=(k-1)*delt; T(k)=t; Y1(k)=ym; Y2(k)=y;
    ys=yset; Ys(k)=ys; I1(k)=s1; I2(k)=s2;
    U1(k)=cont_out1(500);
    U2(k)=cont_out2(500);
    for i=1:499
        cont_out1(i)=cont_out1(i+1);
        cont_out2(i)=cont_out2(i+1);
    end
    s1=s1+(kc1/ti1)*(ys-ym_star)*delt;
    cont_out1(500)=kc1*(ys-ym_star)+s1+kc1*td1*(ys-ym_star-ysb+ymb_star)/delt;
    s2=s2+(kc2/ti2)*(ym-y)*delt;
    cont_out2(500)=kc2*(ym-y)+s2+kc2*td2*(ym-y-ymb+yb)/delt+cont_out1(500);
    ymb=ym; ysb=ys; yb=y; ymb_star=ym_star;
    [x2,y]=compensator_process_ex1(x2,delt,cont_out2);
    [x1,ym]=compensator_model_ex1(x1,delt,cont_out1);
    [x3,ym_star]=compensator_model_star_ex1(x3,delt,cont_out1);
end
figure(1); hold on; plot(T,Ys,T,Y2)
figure(2); hold on; plot(T,U2)

```

```

                                compensator_process_ex1.m
function
[next_x,y]=compensator_
process_ex1(x,delt,u);
subdelt=delt; n=round

```

```

                                compensator_model_ex1.m
function
[next_x,y]=compensator_
model_ex1(x,delt,u);
subdelt=delt; n=round

```

Table 7.3 (Continued)

<pre>(delt/subdelt); A=[0 -1; 1 -2.0]; B=[1.0; 0]; C=[0 1]; delay=1.95; delay_k=round(delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return</pre>	<pre>(delt/subdelt); A=[0 -1; 1 -2.2]; B=[0.95*1.03; 0]; C=[0 1]; delay=1.7*1.03; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return</pre>
<pre>compensator_model_star_ex1 function [next_x,y]=compensator_ model_star_ex1(x,delt,u); subdelt=delt; n=round (delt/subdelt); A=[0 -1; 1 -2.2]; B=[0.95*1.03; 0]; C=[0 1]; delay=0; delay_k=round(delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return</pre>	<pre>command window >> delay_compensator_ex1</pre>

7.3 Gain Scheduling

Gain scheduling is used to incorporate the variation of the process dynamics according to the variation of the operating region. For example, consider the following nonlinear process:

$$\tau \frac{dy(t)}{dt} + y(t) = ku(t - \theta)$$

(7.2)

where $k = k_0 + k_1y(t)$, $\tau = 2.2 + 0.1u(t)$ and θ is constant. Equation (7.2) is a nonlinear FOPTD process. It has a nonlinear static gain of $k = k_0 + k_1y(t)$, which is a function of the process output. Note that the static gain changes (not constant) when the process output moves from one operating region to another operating region, resulting in different dynamic behaviors. Then, how to tune the PID controller? Consider the IMC tuning rule in Section 5.4. This determines the proportional gain by $kk_c = (2 + \theta)/2(\lambda + \theta)$. Then, the proportional gain is $k_c = (2 + \theta)/2k(\lambda + \theta) = (2 + \theta)/2(k_0 + k_1y(t))(\lambda + \theta)$, which changes according to the operating region. This kind of setting is called gain scheduling. The same approach can be applied to a nonlinear time constant. That is, the IMC tuning rule provides the tuning

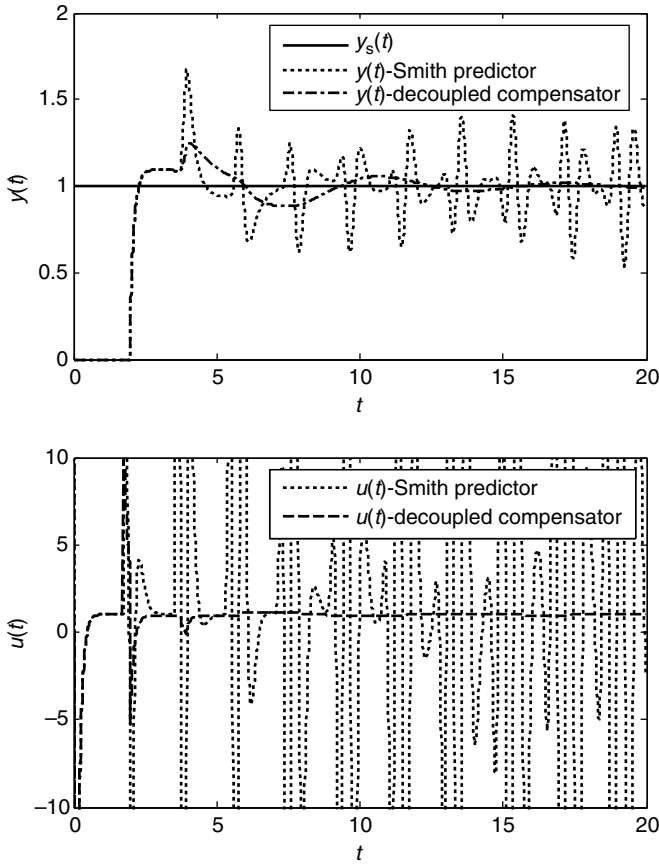


Figure 7.9 Control results of a decoupled time-delay compensator and a Smith predictor.

parameters $\tau_i = \tau + \theta/2 = 2.2 + 0.1u(t) + \theta/2$ because the time constant and the time delay of the process are $2.2 + 0.1u(t)$ and θ . As a result, the PID controller with a gain scheduling of $k_c = [2(2.2 + 0.1u(t)) + \theta]/2(k_0 + k_1y(t))(\lambda + \theta)$, $\tau_i = 2.2 + 0.1u(t) + \theta/2$ should be used to control the nonlinear process of (7.2) to compensate for the nonlinearity of the process.

Gain scheduling can be applied to the case where the nonlinearity of the process is described by a piece-wise function. For example, assume that the process is described by the piece-wise FOPTD model of the following three equations:

$$k = 1, \quad \tau = 1, \quad \theta = 0.2 \quad \text{for } y(t) < 0.2 \quad (7.3)$$

$$k = 1.2, \quad \tau = 0.9, \quad \theta = 0.3 \quad \text{for } 0.2 \leq y(t) < 0.4 \quad (7.4)$$

$$k = 1.5, \quad \tau = 0.7, \quad \theta = 0.35 \quad \text{for } 0.4 \leq y(t) \quad (7.5)$$

Then, the following three tuning parameter sets by the IMC tuning rule with $\lambda = 0.25\theta$ should be used:

$$k_c = 4.4, \quad \tau_i = 1.1, \quad \tau_d = 0.091 \quad \text{for } y(t) < 0.2 \quad (7.6)$$

$$k_c = 2.33, \quad \tau_i = 1.05, \quad \tau_d = 0.129 \quad \text{for } 0.2 \leq y(t) < 0.4 \quad (7.7)$$

$$k_c = 1.373, \quad \tau_i = 0.875, \quad \tau_d = 0.140 \quad \text{for } 0.4 \leq y(t) \quad (7.8)$$

Example 7.4

Simulate the control performance of a PID controller with gain scheduling for the following nonlinear process for the unit step setpoint change:

$$(1 - 0.1y(t)) \frac{dy(t)}{dt} + y(t) = (1 + 0.5y(t))u(t - 0.3) \quad (7.9)$$

Solution In (7.9), the time constant is $\tau = 1 - 0.1y(t)$ and the gain is $k = 1 + 0.5y(t)$. Let us approximate them using the following piece-wise FOPTD model:

$$k = 1.1, \quad \tau = 0.98, \quad \theta = 0.3 \quad \text{for } y(t) < 0.4 \quad (7.10)$$

$$k = 1.3, \quad \tau = 0.94, \quad \theta = 0.3 \quad \text{for } 0.4 \leq y(t) < 0.8 \quad (7.11)$$

$$k = 1.5, \quad \tau = 0.9, \quad \theta = 0.3 \quad \text{for } 0.8 \leq y(t) \quad (7.12)$$

Then, the tuning parameters of the IMC tuning rule for the equations (7.10)–(7.12) are as follows:

$$k_c = 2.739, \quad \tau_i = 1.130, \quad \tau_d = 0.130 \quad \text{for } y(t) < 0.4 \quad (7.13)$$

$$k_c = 2.236, \quad \tau_i = 1.090, \quad \tau_d = 0.129 \quad \text{for } 0.4 \leq y(t) < 0.8 \quad (7.14)$$

$$k_c = 1.867, \quad \tau_i = 1.050, \quad \tau_d = 0.129 \quad \text{for } 0.8 \leq y(t) \quad (7.15)$$

The MATLAB code to simulate the gain scheduling and the simulation results are shown in Table 7.4 and Figure 7.10 respectively. Note that the closed-loop response of PID control with gain scheduling is acceptable, whereas conventional PID control without the gain scheduling shows an oscillatory response. If the nonlinearity is severe, then PID control without gain scheduling may show an unstable closed-loop response.

Table 7.4 MATLAB code to simulate the PID control system with gain scheduling in Example 7.4.

<pre> scheduling_ex1.m clear; t=0.0; t_final=8.0; x=[0]; y=0.0; yb=0.0; ys=0.0; ysb=0.0; delta_t=0.005; n=round (t_final/delta_t); C=[1]; theta=0.3; % time delay h_u=zeros(1,1000); n_theta=round (theta/delta_t); s=0.0; for i=1:n t_array(i)=t; y_array(i)=y; ys_array(i)=ys; if(t>1) ys=1.0; else ys=0.0; end if(y<0.4) kc=2.739; ti=1.130; td=0.130; end if(0.4<=y&y<0.8) kc=2.236; ti=1.090; td=0.129; end if(0.8<=y) kc=1.867; ti=1.050; td=0.129; end % kc=2.739; ti=1.130; td=0.130; % no gain scheduling s=s+(kc/ti)*(ys-y)*delta_t; u=kc*(ys-y)+s+kc*td*((ys-y)- (ysb-yb))/delta_t; ysb=ys; yb=y; % one sampling before u_array(i)=u; for j=1:999 h_u(j)=h_u(j+1); end h_u(1000)=u; dx_dt=g_scheduling_ex1(y,x,h_u (1000-n_theta)); x=x+dx_dt*delta_t; y=C*x; t=t+delta_t; end figure(1); hold on; plot(t_array, ys_array,t_array,y_array); %plot(t_array,y_array); figure(2); hold on; plot(t_array, u_array); </pre>	<pre> g_scheduling_ex1.m function [dx_dt]=g_scheduling_ ex1(y,x,u) A=[-1/(1-0.1*y)]; B=[(1+0.5*y)/(1-0.1*y)]; dx_dt=A*x+B*u; end </pre>
	<pre> command window >> scheduling_ex1 </pre>

7.4 Proportional–Integral–Derivative Control using Internal Feedback Loop

The structure of the PID controller is not appropriate to control an open-loop unstable process such as an integrating or unstable process (Sung and Lee, 1996; Kwak *et al.* 2000). For

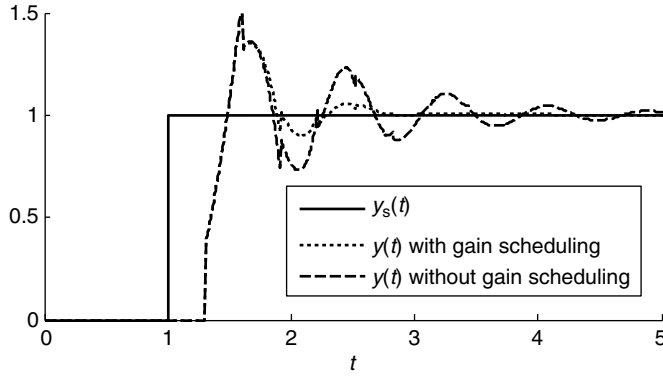


Figure 7.10 Control performances of the PID controller with gain scheduling and without gain scheduling in Example 7.4.

example, consider the following control system composed of the integrating process (7.16) and the PID controller (7.17):

$$G(s) = \frac{\exp(-0.1s)}{s(s+2)} \Rightarrow \frac{d^2y(t)}{dt^2} + 2\frac{dy(t)}{dt} = u(t-0.1) \quad (7.16)$$

$$u(t) = k_c(y_s(t) - y(t)) + \frac{k_c}{\tau_i} \int_0^t (y_s(t) - y(t)) dt + k_c \tau_d \frac{d(y_s(t) - y(t))}{dt} \quad (7.17)$$

From (7.16) and (7.17):

$$\frac{k_c}{\tau_i} \int_0^t (y_s(t) - y(t)) dt = 0.0$$

should be satisfied for the offset to be zero. This is true for all integrating processes. Now, imagine the closed-loop response for a positive step setpoint change. In that case, the final value of $\int_0^\infty (y_s(t) - y(t)) dt$ should be zero. But, the integral term from the starting time $t=0$ and the rise time $t=t_r$ is inevitably a positive value (that is, $\int_0^{t_r} (y_s(t) - y(t)) dt > 0.0$) because perfect control is impossible. So, the integral term from the rise time $t=t_r$ to the final time $t=\infty$ should be negative (that is, $\int_{t_r}^\infty (y_s(t) - y(t)) dt < 0.0$) because

$$\int_0^\infty (y_s(t) - y(t)) dt = \int_0^{t_r} (y_s(t) - y(t)) dt + \int_{t_r}^\infty (y_s(t) - y(t)) dt = 0.0$$

should always be satisfied for an integrating process. This means that a large overshoot (equivalently, a large negative error) cannot be avoided. The same conclusion can be derived for an unstable process. Consider the following control system composed of an unstable process and a PID controller:

$$G(s) = \frac{\exp(-0.1s)}{(s+1)(10s-1)} \Rightarrow 10\frac{d^2y(t)}{dt^2} + 9\frac{dy(t)}{dt} - y(t) = u(t-0.1) \quad (7.18)$$

$$u(t) = k_c(y_s(t) - y(t)) + \frac{k_c}{\tau_i} \int_0^t (y_s(t) - y(t)) dt + k_c \tau_d \frac{d(y_s(t) - y(t))}{dt} \quad (7.19)$$

From (7.18) and (7.19):

$$\frac{k_c}{\tau_i} \int_0^t (y_s(t) - y(t)) dt = -1$$

should be satisfied for the offset to be zero. So, the final value of $\int_0^\infty (y_s(t) - y(t)) dt$ should be negative. Because the integral term at the starting time $t=0$ to the rising time is a positive value (that is, $\int_0^{t_r} (y_s(t) - y(t)) dt > 0.0$), the integral from the rising time to the final time should be negative (that is, $\int_{t_r}^\infty (y_s(t) - y(t)) dt < 0.0$) because

$$\int_0^\infty (y_s(t) - y(t)) dt = \int_0^{t_r} (y_s(t) - y(t)) dt + \int_{t_r}^\infty (y_s(t) - y(t)) dt = -\frac{\tau_i}{k_c}$$

should be satisfied for an unstable process. This means that a large overshoot (equivalently, a large negative error) cannot be avoided. This case is worse than the case of an integrating process because a bigger overshoot is required to satisfy

$$\int_{t_r}^\infty (y_s(t) - y(t)) dt = -\int_0^{t_r} (y_s(t) - y(t)) dt - \frac{\tau_i}{k_c}$$

Until now, the structural limitation of a PID controller in controlling the open-loop unstable process is justified.

Then, how to overcome the structural limitation? This can be solved easily by using an internal feedback control loop (Kwak *et al.*, 2000). Consider the control system of Figure 7.11. The transfer function of the overall process is

$$G_{\text{overall}}(s) = \frac{y(s)}{u(s)} = \frac{G(s)}{1 + G(s)k_i} \quad (7.20)$$

In Figure 7.11, the input and the output of the overall process are $u(s)$ and $y(s)$ respectively. Note that the overall process composed of the open-loop unstable process and the internal feedback loop becomes an open-loop stable process. As a result, there are no more structural limitations.

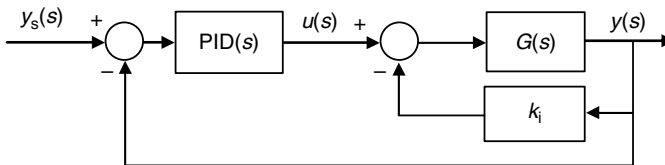


Figure 7.11 PID control using an internal feedback loop.

The PID controller should be tuned on the basis of the overall process through the following procedure. First, the internal feedback loop should be tuned. For the integrating process, k_i can be tuned on the basis of the ultimate gain k_u of the process (for example, $k_i = k_u/4$). The optimal

gain margin tuning rule in Chapter 5 can be used for the unstable process. Second, the PID controller should be tuned. To do that, the overall transfer function of $G_{\text{overall}}(s) = y(s)/u(s) = G(s)/(1 + G(s)k_i)$ should be reduced to an FOPTD or SOPTD model by the model reduction method. Finally, tune the PID controller using the usual tuning rules for the reduced FOPTD or SOPTD model. For a detailed description and examples of the tuning of a PID controller using an internal feedback loop, refer to Chapter 5.

Problems

7.1 Explain the conditions for cascade control to be successful.

7.2 Consider the process of Figure P7.1. Determine if the cascade control is recommendable to the following cases. Here, $y_1(t)$ is measurable.

- (a) $G_{p2}(s) = \frac{\exp(-0.1s)}{s+1}$, $G_{p1}(s) = \frac{\exp(-0.3s)}{(s+1)^5}$, $y_2(t)$ is measurable
- (b) $G_{p2}(s) = \frac{\exp(-0.8s)}{s+1}$, $G_{p1}(s) = \frac{\exp(-0.1s)}{(s+1)^2}$, $y_2(t)$ is measurable
- (c) $G_{p2}(s) = \frac{\exp(-0.8s)}{s+1}$, $G_{p1}(s) = \frac{\exp(-0.1s)}{(s+1)^2}$, $y_2(t)$ is not measurable
- (d) $G_{p2}(s) = \frac{10\exp(-0.5s)}{(s+1)^2}$, $G_{p1}(s) = \frac{\exp(-0.5s)}{(s+1)^2}$, $y_2(t)$ is measurable.

7.3 Design a cascade control system for the process in Figure P7.1 with $G_{p2}(s) = \exp(-0.1s)/(s+1)^2$ and $G_{p1}(s) = \exp(-1.0s)/(s+1)^3$ and simulate the control performance for a step input disturbance.

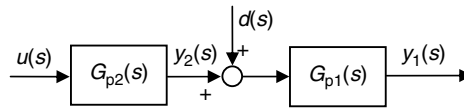


Figure P7.1

7.4 Design a Smith predictor for the process $G(s) = \exp(-1.5s)/(s+1)^3$ and simulate the control performance for a step setpoint change.

7.5 Design the gain scheduling for the following nonlinear process and simulate the control performance for a step setpoint change:

- (a) $(1 + 0.2y(t)) \frac{dy(t)}{dt} + y(t) = (1 + 1.5y(t))u(t - 0.5)$
- (b) $(1 + 0.1y(t))^2 \frac{d^2y(t)}{dt^2} + 2(1 + 0.1y(t))(1 + 0.5y(t)) \frac{dy(t)}{dt} + y(t) = (2.0 + u(t - 0.5))u(t - 0.5)$

7.6 Summarize the advantages of a Smith predictor and gain scheduling.

7.7 Design the decoupled time-delay compensator for the process and the model of $G_m^*(s) = G_m(s) = G_p(s) = \exp(-0.2s)/(s+1)^4$ and simulate the control performances

for a step setpoint change and a step input disturbance rejection. In this case, it is not a time-delay compensator, but it must show good control performances for both the step setpoint change and the step input disturbance rejection.

- 7.8 Design the decoupled time-delay compensator for the process $G_p(s) = \exp(-1.0s)/(s + 1)^4$ and simulate the control performances for a step setpoint change. In this case, reduce the process using the model reduction method to obtain the SOPTD model.
- 7.9 Design a PID controller and internal feedback loop for the process $G_m(s) = G_p(s) = \exp(-1.5s)/s(s + 1)^2$ and simulate the control performances for a step setpoint change and a step input disturbance rejection.

References

- Kwak, H.J., Sung, S.W. and Lee, I. (2000) Stabilizability conditions and controller design for unstable processes. *Chemical Engineering Research & Design*, **78**, 549.
- Lee, D., Lee, M., Sung, S.W. and Lee, I. (1999) Robust PID tuning for Smith predictor in the presence of model uncertainty. *Journal of Process Control*, **9**, 79.
- Smith, O.J.M. (1957) Closer control of loops with dead time. *Chemical Engineering Progress*, **53**, 217.
- Sung, S.W. and Lee, I. (1996) Limitations and countermeasures of PID controllers. *Industrial & Engineering Chemistry Research*, **35**, 2596.

Bibliography

- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
- Stephanopoulos, G. (1984) *Chemical Process Control – An Introduction to Theory and Practice*, Prentice-Hall.

Part Three

Process Identification

Process identification methods, whose role is to provide the process model in designing the process controller, are introduced in Part Three. In Chapter 8, the mathematical tools of the Fourier series and describing function analysis are introduced, followed by the process identification methods of the Fourier analysis and the modified Fourier transform to estimate the process models in the form of the frequency response. Chapters 9 and 10 introduce the process identification methods used to obtain the process models in the form of a continuous-time differential equation and a discrete-time difference equation respectively. Chapter 11 discusses how to convert the discrete-time model to a continuous-time model.

8

Process Identification Methods for Frequency Response Models

A Fourier series is one of the most important representations for describing a periodic function. The Fourier series and Fourier transform have been widely used to identify process models. This chapter introduces several process identification methods to estimate the frequency response data of the process.

8.1 Fourier Series

The Fourier series is an important basic theory needed in deriving and analyzing process identification methods. In this section, the formulas to calculate the Fourier coefficients of the Fourier series are derived.

Assume that the periodic function has a period p . It is proven that all the data of the periodic function can be represented by the following Fourier series (Kreyszig, 2006):

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2n\pi t}{p}\right) + b_n \sin\left(\frac{2n\pi t}{p}\right) \right] \quad (8.1)$$

where the coefficients are called the Fourier coefficients. Then, how to calculate the Fourier coefficients for the given periodic function? Let us derive the formula.

Formula to obtain a_0 Let us integrate both sides of (8.1) from $t = -p/2$ to $t = p/2$. Then, (8.2) is obtained:

$$\int_{-p/2}^{p/2} f(t) dt = \int_{-p/2}^{p/2} a_0 dt + \sum_{n=1}^{\infty} \left[a_n \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) dt + b_n \int_{-p/2}^{p/2} \sin\left(\frac{2n\pi t}{p}\right) dt \right] \quad (8.2)$$

It is straightforward to derive (8.3) from (8.2):

$$\int_{p/2}^{p/2} f(t) dt = pa_0 + \sum_{n=1}^{\infty} (a_n \times 0 + b_n \times 0) \rightarrow a_0 = \frac{1}{p} \int_{-p/2}^{p/2} f(t) dt \quad (8.3)$$

Formula to obtain $a_m, m = 1, 2, \dots$. Let us integrate both sides of (8.1) from $t = -p/2$ to $t = p/2$ after multiplying by $\cos(2m\pi t/p)$. Then, (8.4) is obtained:

$$\begin{aligned} \int_{-p/2}^{p/2} f(t) \cos\left(\frac{2m\pi t}{p}\right) dt &= \int_{-p/2}^{p/2} a_0 \cos\left(\frac{2m\pi t}{p}\right) dt + \sum_{n=1}^{\infty} \left[a_n \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) \cos\left(\frac{2m\pi t}{p}\right) dt \right. \\ &\quad \left. + b_n \int_{-p/2}^{p/2} \sin\left(\frac{2n\pi t}{p}\right) \cos\left(\frac{2m\pi t}{p}\right) dt \right] \end{aligned} \quad (8.4)$$

Here, consider the following:

$$\int_{-p/2}^{p/2} a_0 \cos\left(\frac{2m\pi t}{p}\right) dt = 0 \quad (8.5)$$

$$\int_{-p/2}^{p/2} \sin\left(\frac{2n\pi t}{p}\right) \cos\left(\frac{2m\pi t}{p}\right) dt = \frac{1}{2} \int_{-p/2}^{p/2} \sin\left[\frac{2(n+m)\pi t}{p}\right] dt + \frac{1}{2} \int_{-p/2}^{p/2} \sin\left[\frac{2(n-m)\pi t}{p}\right] dt = 0 \quad (8.6)$$

$$\begin{aligned} \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) \cos\left(\frac{2m\pi t}{p}\right) dt \\ = \int_{-p/2}^{p/2} \cos^2\left(\frac{2m\pi t}{p}\right) dt = \frac{1}{2} \left[\int_{-p/2}^{p/2} 1 dt + \int_{-p/2}^{p/2} \cos\left(\frac{4m\pi t}{p}\right) dt \right] = \frac{p}{2} \quad \text{for } n = m \end{aligned} \quad (8.7)$$

$$\begin{aligned} \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) \cos\left(\frac{2m\pi t}{p}\right) dt \\ = \frac{1}{2} \int_{-p/2}^{p/2} \cos\left[\frac{2(n+m)\pi t}{p}\right] dt + \frac{1}{2} \int_{-p/2}^{p/2} \cos\left[\frac{2(n-m)\pi t}{p}\right] dt = 0 \quad \text{for } n \neq m \end{aligned} \quad (8.8)$$

So, (8.9) is obtained from (8.4):

$$\int_{-p/2}^{p/2} f(t) \cos\left(\frac{2m\pi t}{p}\right) dt = \frac{pa_m}{2} \rightarrow a_m = \frac{2}{p} \int_{-p/2}^{p/2} f(t) \cos\left(\frac{2m\pi t}{p}\right) dt \quad (8.9)$$

Formula to obtain $b_m, m = 1, 2, \dots$. Let us integrate both sides of (8.1) from $t = -p/2$ to $t = p/2$ after multiplying by $\sin(2m\pi t/p)$. Then, (8.10) is obtained:

$$\begin{aligned} & \int_{-p/2}^{p/2} f(t) \sin\left(\frac{2m\pi t}{p}\right) dt \\ &= \int_{-p/2}^{p/2} a_0 \sin\left(\frac{2m\pi t}{p}\right) dt + \sum_{n=1}^{\infty} \left[a_n \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) \sin\left(\frac{2m\pi t}{p}\right) dt \right. \\ & \quad \left. + b_n \int_{-p/2}^{p/2} \sin\left(\frac{2n\pi t}{p}\right) \sin\left(\frac{2m\pi t}{p}\right) dt \right] \end{aligned} \quad (8.10)$$

Here, consider the following:

$$\int_{-p/2}^{p/2} a_0 \sin\left(\frac{2m\pi t}{p}\right) dt = 0 \quad (8.11)$$

$$\begin{aligned} & \int_{-p/2}^{p/2} \cos\left(\frac{2n\pi t}{p}\right) \sin\left(\frac{2m\pi t}{p}\right) dt \\ &= \frac{1}{2} \int_{-p/2}^{p/2} \sin\left[\frac{2(n+m)\pi t}{p}\right] dt + \frac{1}{2} \int_{-p/2}^{p/2} \sin\left[\frac{2(n-m)\pi t}{p}\right] dt = 0 \end{aligned} \quad (8.12)$$

$$\begin{aligned} & \int_{-p/2}^{p/2} \sin\left(\frac{2m\pi t}{p}\right) \sin\left(\frac{2m\pi t}{p}\right) dt \\ &= \int_{-p/2}^{p/2} \sin^2\left(\frac{2m\pi t}{p}\right) dt = \frac{1}{2} \left[\int_{-p/2}^{p/2} 1 dt - \int_{-p/2}^{p/2} \cos\left(\frac{2m\pi t}{p}\right) dt \right] = \frac{p}{2} \quad \text{for } n = m \end{aligned} \quad (8.13)$$

$$\begin{aligned} & \int_{-p/2}^{p/2} \sin\left(\frac{2n\pi t}{p}\right) \sin\left(\frac{2m\pi t}{p}\right) dt \\ &= \frac{1}{2} \int_{-p/2}^{p/2} \cos\left[\frac{2(n-m)\pi t}{p}\right] dt - \frac{1}{2} \int_{-p/2}^{p/2} \cos\left[\frac{2(n+m)\pi t}{p}\right] dt = 0 \quad \text{for } n \neq m \end{aligned} \quad (8.14)$$

So, (8.15) is obtained from (8.10):

$$\int_{-p/2}^{p/2} f(t) \sin\left(\frac{2m\pi t}{p}\right) dt = \frac{pb_m}{2} \rightarrow b_m = \frac{2}{p} \int_{-p/2}^{p/2} f(t) \sin\left(\frac{2m\pi t}{p}\right) dt \quad (8.15)$$

In summary, a periodic signal $f(t)$ of which the period is p can be represented by a Fourier series:

$$f(t) = a_0 + \sum_{n=1}^{\infty} \left[a_n \cos\left(\frac{2n\pi t}{p}\right) + b_n \sin\left(\frac{2n\pi t}{p}\right) \right] \quad (8.16)$$

where the coefficients of the Fourier series are estimated by the following formulas:

$$a_0 = \frac{1}{p} \int_{-p/2}^{p/2} f(t) dt = \frac{1}{p} \int_0^p f(t) dt \quad (8.17)$$

$$a_n = \frac{2}{p} \int_{-p/2}^{p/2} f(t) \cos\left(\frac{2n\pi t}{p}\right) dt = \frac{2}{p} \int_0^p f(t) \cos\left(\frac{2n\pi t}{p}\right) dt, \quad n = 1, 2, \dots \quad (8.18)$$

$$b_n = \frac{2}{p} \int_{-p/2}^{p/2} f(t) \sin\left(\frac{2n\pi t}{p}\right) dt = \frac{2}{p} \int_0^p f(t) \sin\left(\frac{2n\pi t}{p}\right) dt, \quad n = 1, 2, \dots \quad (8.19)$$

Example 8.1

Represent the periodic signal shown in Figure 8.1 using a Fourier series.

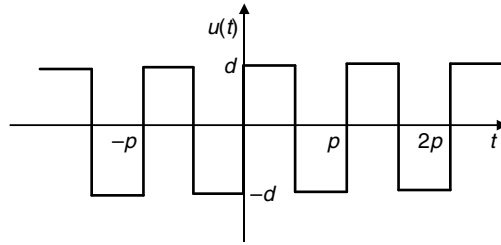


Figure 8.1 A periodic signal for which the period is p .

Solution Because $u(t)$ is a periodic signal, it can be represented by the Fourier series (8.20) as follows:

$$u(t) = a_0 + a_1 \cos\left(\frac{2\pi t}{p}\right) + b_1 \sin\left(\frac{2\pi t}{p}\right) + a_2 \cos\left(2 \times \frac{2\pi t}{p}\right) + b_2 \sin\left(2 \times \frac{2\pi t}{p}\right) + \dots \quad (8.20)$$

Because $u(t)$ is an odd function, then

$$a_0 = \frac{1}{p} \int_0^p u(t) dt = 0, \quad a_n = \frac{2}{p} \int_0^p u(t) \cos\left(\frac{2n\pi t}{p}\right) dt = 0$$

So, the Fourier series becomes

$$u(t) = b_1 \sin\left(\frac{2\pi t}{p}\right) + b_2 \sin\left(2 \times \frac{2\pi t}{p}\right) + b_3 \sin\left(3 \times \frac{2\pi t}{p}\right) + \dots \quad (8.21)$$

The Fourier coefficients in (8.21) can be obtained by (8.19) as follows:

$$b_n = \frac{2}{p} \int_0^p u(t) \sin\left(\frac{2n\pi t}{p}\right) dt = \frac{2}{p} \int_0^{p/2} d \sin\left(\frac{2n\pi t}{p}\right) dt - \frac{2}{p} \int_{p/2}^p d \sin\left(\frac{2n\pi t}{p}\right) dt \quad (8.22)$$

Equation (8.22) can be rewritten as

$$b_n = \frac{4}{p} \int_0^{p/2} d \sin\left(\frac{2n\pi t}{p}\right) dt \quad (8.23)$$

Thus:

$$b_n = \begin{cases} -\frac{4d}{2n\pi} \cos\left(\frac{2n\pi t}{p}\right) \Big|_0^{p/2} = -\frac{4d}{2n\pi} [\cos(n\pi) - \cos(0)] = \frac{4d}{n\pi}, & n = 1, 3, 5, \dots \\ -\frac{4d}{2n\pi} \cos\left(\frac{2n\pi t}{p}\right) \Big|_0^{p/2} = -\frac{4d}{2n\pi} [\cos(n\pi) - \cos(0)] = 0, & n = 2, 4, 6, \dots \end{cases} \quad (8.24)$$

So, the final form of $u(t)$ is

$$u(t) = \frac{4d}{\pi} \sin(\omega t) + \frac{4d}{3\pi} \sin(3\omega t) + \frac{4d}{5\pi} \sin(5\omega t) + \frac{4d}{7\pi} \sin(7\omega t) + \dots \quad (8.25)$$

where $\omega = 2\pi/p$ is the fundamental frequency of $u(t)$. Figure 8.2 compares the original function of $u(t)$ and the approximated function by the finite Fourier series of

$$u_n(t) = \frac{4d}{\pi} \sin\left(\frac{2\pi}{p} t\right) + \frac{4d}{3\pi} \sin\left(3 \frac{2\pi}{p} t\right) + \dots + \frac{4d}{n\pi} \sin\left(n \frac{2\pi}{p} t\right)$$

As the number of the terms increases, a better accuracy is obtained.

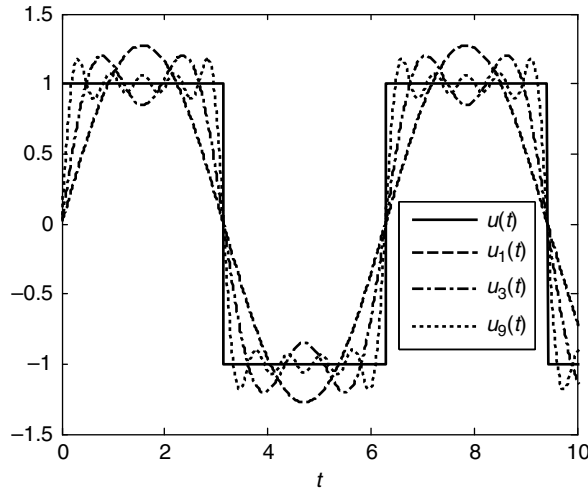


Figure 8.2 Approximation results of a finite Fourier series.

8.2 Frequency Response Analysis and Autotuning

Let us explain the conventional relay feedback method briefly before introducing frequency response analysis and autotuning. Figure 8.3 shows an activated process output by a conventional relay feedback method.

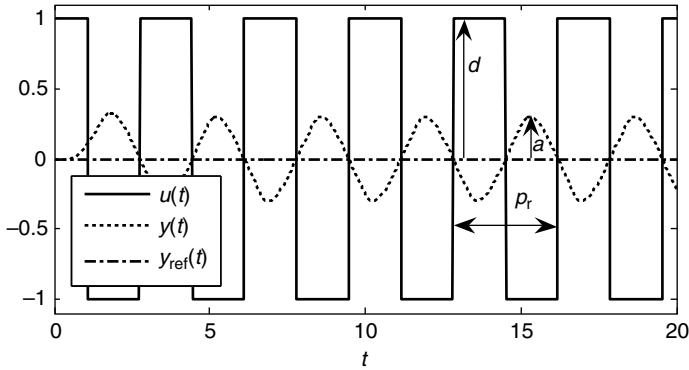


Figure 8.3 Activated process output by a conventional relay feedback method.

The procedure for process activation by relay feedback is as follows. First, the upper (on) value of the relay output is applied to drag the process output out of the initial value, as shown in Figure 8.3. Second, the lower (off) value of the relay is applied when the process output deviates from the initial state. Third, the upper value of the relay is applied when the process output is less than the reference value, and vice versa. That is, $u(t) = d$ if $y(t) \leq 0$ and $u(t) = -d$ if $y(t) > 0$. Then, the process input and output usually reach a cyclic steady state (which means that the period and the peak value of the process output do not change) after three or four cycles. For a more detailed description on the relays, refer to Chapter 12.

8.2.1 Frequency Response Analysis

The objective of frequency response analysis is to estimate the frequency response (frequency model) of the process from the activated process input and the process output. For example, consider activated process data from a conventional relay feedback method, as shown in Figure 8.3. To estimate the frequency model of the process from the activated process input and output, the two signals of the relay output and the process output are approximated to two sine signals. As shown in (8.25), the square signal of the relay output in the cyclic steady state can be represented by a Fourier series as follows:

$$u(t) = \frac{4d}{\pi} \sin(\omega t) + \frac{4d}{3\pi} \sin(3\omega t) + \frac{4d}{5\pi} \sin(5\omega t) + \dots \quad (8.26)$$

where $\omega = 2\pi/p_r$ is the fundamental frequency. p_r denotes the period of the relay. $u(t)$ and d denote the relay output and the magnitude of the relay on–off respectively. If only the fundamental term is considered and the harmonics (higher frequency terms such as

$4d \sin(3\omega t)/3\pi, 4d \sin(5\omega t)/5\pi, \dots$ are neglected, then the following approximation is obtained:

$$u(t) \approx \frac{4d}{\pi} \sin(\omega t) \quad (8.27)$$

Also, a sine signal can approximate the process output as follows:

$$y(t) \approx -a \sin(\omega t) = a \sin(\omega t - \pi) \quad (8.28)$$

where $y(t)$ and a denote the process output and the peak value of the process output respectively. Then, the process output can be said to be approximately $y(t) \approx a \sin(\omega t - \pi)$ for the process input $u(t) \approx 4d \sin(\omega t)/\pi$. Then, it is clear that the phase angle between $u(t)$ and $y(t)$ is $-\pi$. So, the frequency ω is the ultimate frequency of the process. Also, it is clear that the amplitude ratio corresponding to the ultimate frequency ω is approximately $\pi a/4d$ and the ultimate gain is the reciprocal of $\pi a/4d$. In summary:

$$\omega_u \approx \omega = \frac{2\pi}{p_r} \quad \text{and} \quad p_u \approx p_r \quad (8.29)$$

$$k_u \approx \frac{4d}{\pi a} \quad (8.30)$$

where ω_u , p_u and k_u denote the ultimate frequency, the ultimate period and the ultimate gain respectively.

8.2.2 Autotuning

Autotuning is tuning the PID controller in an automatic way. It goes through the following steps. Step 1 (process activation), activate the process using the relay feedback method as shown in Figure 8.3. Step 2 (modeling), estimate the ultimate period and the ultimate gain using (8.29) and (8.30) from the measured period, the measured peak value of the process output and the magnitude of the relay output. Step 3 (tuning), calculate the tuning parameters of the PID controller using the ZN tuning rule. Step 4 (downloading), download the tuning parameters to the PID controller. When the user sends a signal to the autotuner by pushing a start button, the autotuner performs the whole procedure from Step 1 to Step 4 automatically. So, the user who has no knowledge of process control can successfully tune a PID controller in a very simple way.

Example 8.2

Simulate Figure 8.3 with the process $G(s) = \exp(-0.5s)/(s + 1)^2$.

Solution The MATLAB code to simulate Figure 8.3 is shown in Table 8.1.

8.3 Describing Function Analysis

Describing function analysis can be used to derive (8.29) and (8.30) (Åström and Hägglund, 1984, 1995). A describing function is a transfer function of a nonlinear element for a given frequency. For example, consider the symbol representing the ideal relay on–off in Figure 8.4, where the x -axis and the y -axis represent the input of the relay and the output of the

Table 8.1 MATLAB code to simulate Figure 8.3 with the process $G(s) = \exp(-0.5s)/(s + 1)^2$.

<pre> conv_relay_ex1.m clear; delt=0.02; tf=20; n=round(tf/delt); x=zeros(2,1); u_data=zeros(1,500); t_on=0.0; t_off=0.0; P_on=0; P_off=0; ymin=0.0; ymax=0.0; y=0.0; yref=0.0; index=0; y_delta=0.1; % initial phase:index=0, relay phase:index=1 for i=1:n t=i*delt; yy(i)=y; yyref(i)=yref; tt(i)=t; if(index==1) if(yy(i)>yref & yy(i-1)<=yref) P_on=t-t_on; t_off=t; ymax_f=ymax; ymax=0.0; end if(yy(i)<=yref & yy(i-1)>yref) P_off=t-t_off; t_on=t; ymin_f=ymin; ymin=0.0; end end if(y>yref) u=-1.0; if(y>ymax) ymax=y; end end if(y<=yref) u=1.0; if(y<ymin) ymin=y; end end if(index==0) u=1.0; if(y>y_delta) index=1; end end for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u; uu(i)=u; [x,y]=g_conv_relay_ex1(x,delt,u_data); end P=P_on+P_off; a=(abs(ymax_f)+abs(ymin_f))/2; AR_u=a/(4/pi); w_u=2*pi/P %ultimate frequency P_u=P %ultimate period AR_u %ultimate gain figure(1); plot(tt,uu,tt,yy,tt,yyref); </pre>	<pre> g_conv_relay_ex1.m function [next_x,y]=g_conv_relay_ex1 (x,delt,u); subdelt=delt; n=round (delt/subdelt); A=[0 -1;1 -2]; B=[1;0]; C=[0 1]; delay=0.5; delay_k=round (delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return end command window >> conv_relay_ex1 w_u = 1.8700 P_u = 3.3600 AR_u = 0.2354 Kc_u = 4.2488 </pre>
--	---

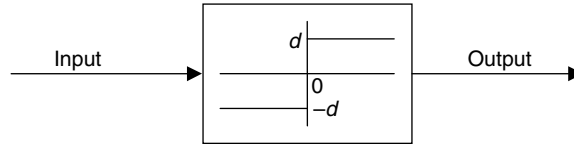


Figure 8.4 Symbol of an ideal relay.

relay respectively. So, if the input is greater than zero, then the relay output is d . Otherwise, the relay output is $-d$. Assume that the input is the sine signal of $a \sin(\omega t)$, where a and ω are respectively the amplitude and the frequency of the sine signal. Then, the relay output will be the square signal in Figure 8.5 according to the relay symbol in Figure 8.4.

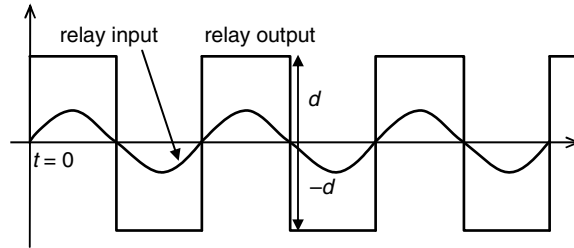


Figure 8.5 Relay output for a sine input.

The square signal of the relay output can be represented by a Fourier series and it can be approximated by a sine signal of the fundamental frequency as follows:

$$u(t) = \frac{4d}{\pi} \sin(\omega t) + \frac{4d}{3\pi} \sin(3\omega t) + \frac{4d}{5\pi} \sin(5\omega t) + \dots \approx \frac{4d}{\pi} \sin(\omega t) \quad (8.31)$$

where $u(t)$, d and ω denote the relay output, the magnitude of the relay on-off and the relay frequency respectively. p_r denotes the period of the relay. Then, when entering the $a \sin(\omega t)$ signal into the nonlinear element (here, the relay) the output of the nonlinear element is $4d \sin(\omega t)/\pi$. So, the amplitude ratio of the relay is $4d/\pi a$ and the phase angle is zero. So, the transfer function (the describing function) of the relay is

$$N(a) = \frac{4d}{\pi a} \exp(-0i) = \frac{4d}{\pi a} \quad (8.32)$$

Until now the describing function of the ideal relay has been derived. Again, it is emphasized that the describing function is just a transfer function of the nonlinear element for a given frequency.

Now, consider the block diagram in Figure 8.6 for relay feedback control. Note that it produces the same oscillation as that in Figure 8.3 in the cyclic steady state because the input of the relay is a negative feedback of the process output (that is, $-y(t)$). Because the closed-loop system shows continuous cycling (marginally stable), the characteristic equation of Figure 8.6 will satisfy the following condition:

$$1 + N(a)G(i\omega) = 0 \quad (8.33)$$

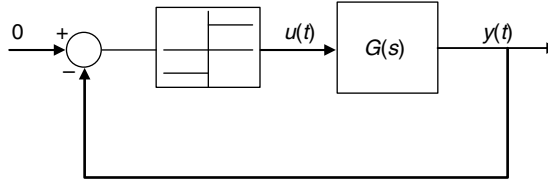


Figure 8.6 Block diagram of an ideal relay feedback control system to activate the process output.

So, the frequency information of the process for the relay frequency $\omega = 2\pi/p_r$ can be estimated from (8.33) as follows:

$$G(i\omega) = -\frac{1}{N(a)} = -\frac{\pi a}{4d} \quad (8.34)$$

Here, it should be noted that the imaginary part of $G(i\omega)$ is zero. So, the frequency $\omega = 2\pi/p_r$ is the ultimate frequency of the process and the reciprocal of the absolute value of $G(i\omega)$ is the ultimate gain of the process. That is:

$$\omega_u \approx \omega = \frac{2\pi}{p_r} \quad \text{and} \quad p_u \approx p_r \quad (8.35)$$

$$k_u \approx \frac{4d}{\pi a} \quad (8.36)$$

where ω_u , p_u and k_u denote the ultimate frequency, the ultimate period and the ultimate gain respectively. These results are the same as (8.29) and (8.30).

Example 8.3

Obtain the describing function for the nonlinear element in Figure 8.7. This is composed of two channels (Friman and Waller, 1997). One is the proportional channel of the conventional relay and the other is the integral channel of the conventional relay combined with the integrator. Here, the magnitude of the relay is one.

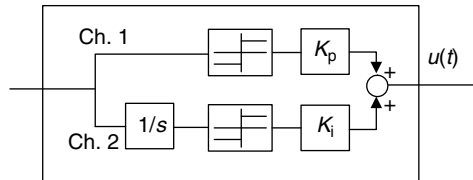


Figure 8.7 Two-channel relay composed of conventional relays and an integrator.

Solution Assume that the input of the two-channel relay is the sine signal of $a \sin(\omega t)$. Then, the output of the proportional channel is the square signal in Figure 8.5 according to the relay symbol in Figure 8.7. Then, the output of the proportional channel is approximately

$K_p 4 \sin(\omega t)/\pi$. So, the describing function of the proportional channel is the same as that of the ideal relay:

$$N(a, K_p) = K_p \frac{4}{\pi a} \quad (8.37)$$

Consider the integral channel. The sine signal $a \sin(\omega t)$ goes through the integrator. Then, it becomes $-a \cos(\omega t)/\omega = a \sin(\omega t - \pi/2)/\omega$. Then, the relay input of the integral channel is lagged by $\pi/2$ compared with the input sine signal. This means that the output of the integral channel is $K_i 4 \sin(\omega t - \pi/2)/\pi$ approximately. So, the amplitude and the phase angle of the integral channel are $K_i 4/\pi a$ and $-\pi/2$ respectively. Equivalently, the describing function of the integral channel is

$$N(a, K_i) = K_i \frac{4}{\pi a} \exp(-i\pi/2) \quad (8.38)$$

Finally, the overall describing function of the two-channel relay is

$$\begin{aligned} N(a, K_p, K_i) &= \frac{4}{\pi a} [K_p + K_i \exp(-i\pi/2)] = \frac{4}{\pi a} (K_p - iK_i) \\ &= \frac{4}{\pi a} \sqrt{K_p^2 + K_i^2} \exp[-i \arctan(K_i/K_p)] \end{aligned} \quad (8.39)$$

Example 8.4

Obtain the frequency response of the process from the process input and output activated by the two-channel relay in Example 8.3. Here, the magnitude of the oscillation of the activated process output is a and the period of the oscillation is p_r . That is, the process output is approximately $a \sin(\omega t)$, $\omega = 2\pi/p_r$.

Solution Because the process output is in continuous cycling, the following equation is satisfied:

$$1 + N(a, K_p, K_i)G(i\omega) = 0 \quad (8.40)$$

Then, the frequency response of the process at the frequency ω is

$$G(i\omega) = -\frac{1}{N(a, K_p, K_i)} = -\frac{\pi a}{4\sqrt{K_p^2 + K_i^2}} \exp[i \arctan(K_i/K_p)] \quad (8.41)$$

It should be noted that the real and the imaginary parts are both negative. So, the frequency region identified by (8.41) is a lower frequency region than the ultimate frequency, as shown in Figure 8.8. As a result, the two-channel relay feedback method can identify the frequency response data of the process in the third quadrant of the Nyquist plot by adjusting K_p and K_i .

Example 8.5

Obtain the describing function for the following nonlinear element. It is the ideal relay combined with the time delay (Kim, 1995; Tan *et al.*, 1996). Here, the magnitude of the relay is d .

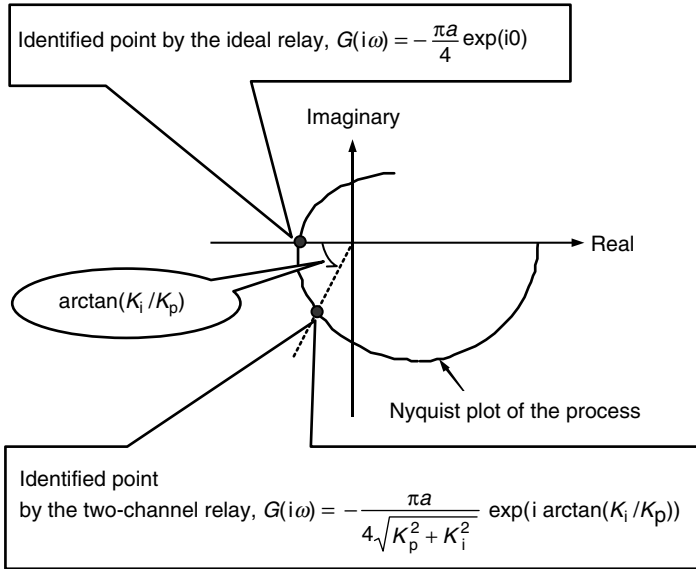


Figure 8.8 Identified Nyquist point by the ideal unbiased-relay feedback method and the two-channel relay feedback method.

Solution Assume that the input of the relay combined with the time delay is the sine signal $a \sin(\omega t)$. Then, the output of the nonlinear element in Figure 8.9 is approximately $u(t) = 4d \sin(\omega t - \omega\theta)/\pi$ because the square signal of the ideal relay is delayed as long as θ . Then, the amplitude ratio and the phase angle are $4d/\pi a$ and $-\omega\theta$ respectively. Equivalently, the describing function of the relay plus time delay is

$$N(a, \theta) = \frac{4d}{\pi a} \exp(-i\omega\theta) \quad (8.42)$$

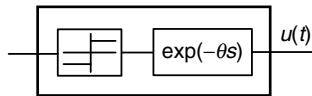


Figure 8.9 Relay combined with time delay.

Example 8.6

Obtain the frequency response of the process from the process input and output activated by the relay plus time delay in Example 8.5. Here, the magnitude of the oscillation of the activated process output is a and the period of the oscillation is p_r . That is, the process output is approximately $a \sin(\omega t)$, $\omega = 2\pi/p_r$.

Solution Because the process output is in continuous cycling, the following equation is satisfied:

$$1 + N(a, \theta)G(i\omega) = 0 \quad (8.43)$$

Then, the frequency response of the process at the frequency ω is

$$G(i\omega) = -\frac{1}{N(a, \theta)} = -\frac{\pi a}{4d} \exp(i\omega\theta) \quad (8.44)$$

The real and the imaginary parts are both negative. So, the frequency region identified by (8.44) is a lower frequency region than the ultimate frequency.

8.4 Fourier Analysis

The describing function analysis cannot provide exact frequency data of the process because it approximates a square signal to a sine signal. Fourier analysis can overcome this problem. It can estimate the frequency response data of the process without the modeling error from the process input and the process output activated by the relay (Sung and Lee, 1997).

8.4.1 Fourier Analysis

Assume that the process input (that is, relay output) $u(t)$ and the process output $y(t)$ after t_{ss} are periodic with the period of p_r . Then, the frequency response of the process can be estimated by

$$G(i\omega) = \frac{\int_{t_{ss}}^{t_{ss}+p_r} y(t) \exp(-i\omega t) dt}{\int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i\omega t) dt} \quad (8.45)$$

where ω should be zero or a multiple of $\omega_r = 2\pi/p_r$. That is, the frequency responses of the process for several frequencies (zero, $\omega_r = 2\pi/p_r$, $2\omega_r$, $3\omega_r$, ...) can be estimated using (8.45) with numerical integration.

8.4.2 Derivation of Fourier Analysis

From the transfer function

$$G(s) = \frac{\int_0^\infty y(t) \exp(-st) dt}{\int_0^\infty u(t) \exp(-st) dt}$$

(8.46) is obtained:

$$G(i\omega) \equiv \frac{\int_0^\infty y(t) \exp(-i\omega t) dt}{\int_0^\infty u(t) \exp(-i\omega t) dt} \quad (8.46)$$

Now, consider (8.47).

$$\begin{aligned} \int_0^\infty y(t) \exp(-i\omega t) dt &= \int_0^{t_{ss}} y(t) \exp(-i\omega t) dt + \int_{t_{ss}}^{t_{ss}+p_r} y(t) \exp(-i\omega t) dt \\ &\quad + \int_{t_{ss}+p_r}^{t_{ss}+2p_r} y(t) \exp(-i\omega t) dt + \dots \end{aligned} \quad (8.47)$$

Note that $\exp(-i\omega t) = \cos(\omega t) - i \sin(\omega t)$ is a periodic function of period p_r because ω is zero or the multiple of $\omega_r = 2\pi/p_r$. Also, $u(t)$ and $y(t)$ after t_{ss} are also periodic functions of the same period. Then, $y(t)\exp(-i\omega t)$ is also a periodic function of the period of p_r . Then, (8.48) and (8.49) are derived:

$$\int_0^{\infty} y(t)\exp(-i\omega t) dt = \int_0^{t_{ss}} y(t)\exp(-i\omega t) dt + \lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} y(t)\exp(-i\omega t) dt \quad (8.48)$$

$$\int_0^{\infty} u(t)\exp(-i\omega t) dt = \int_0^{t_{ss}} u(t)\exp(-i\omega t) dt + \lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} u(t)\exp(-i\omega t) dt \quad (8.49)$$

Also, note that $\int_0^{t_{ss}} y(t)\exp(-i\omega t) dt$ and $\int_0^{t_{ss}} u(t)\exp(-i\omega t) dt$ are negligible compared with $\lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} y(t)\exp(-i\omega t) dt$ and $\lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} u(t)\exp(-i\omega t) dt$. So, the Fourier analysis (8.50) can be derived from (8.46), (8.48), and (8.49):

$$G(i\omega) \equiv \frac{\int_0^{t_{ss}} y(t)\exp(-i\omega t) dt + \lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} y(t)\exp(-i\omega t) dt}{\int_0^{t_{ss}} u(t)\exp(-i\omega t) dt + \lim_{n_p \rightarrow \infty} n_p \int_{t_{ss}}^{t_{ss}+p_r} u(t)\exp(-i\omega t) dt} = \frac{\int_{t_{ss}}^{t_{ss}+p_r} y(t)\exp(-i\omega t) dt}{\int_{t_{ss}}^{t_{ss}+p_r} u(t)\exp(-i\omega t) dt} \quad (8.50)$$

No approximations are used for the derivation of (8.50). So, the frequency data obtained using (8.45) are exact only if the signals of $u(t)$ and $y(t)$ are periodic functions after t_{ss} . Usually, the activated signals by the relay feedback become periodic functions after three or four relay on–offs. Then, all the frequency responses corresponding to the multiples of the relay frequency and zero frequency can be exactly identified by (8.45). For example, the three lots of frequency response data of the process for 0, ω_r and $2\omega_r$ can be estimated by calculating (8.45) repetitively for 0, ω_r and $2\omega_r$ only if the activated signals include the three frequency components.

8.4.3 Application of Fourier Analysis

Consider the process input and output in Figure 8.10 activated by a biased-relay feedback method. In biased-relay feedback, the reference value for the relay on–off is a bias of 0.5 rather

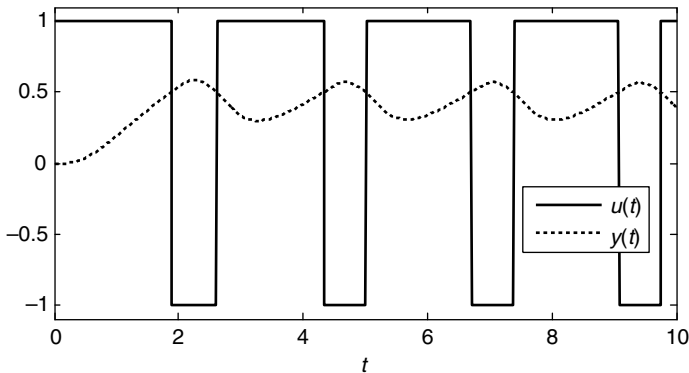


Figure 8.10 Process activation by the biased-relay feedback method.

than zero (zero means the initial process output value). That is, the relay output is 1.0 if the process output is greater than the reference value (bias) of 0.5 and the relay output is -1.0 if the process output is less than 0.5. For a more detailed description on relays, refer to Chapter 12. It is clear in Figure 8.10 that $\int_{t_{ss}}^{t_{ss}+p_r} y(t) \exp(-i\omega t) dt$ and $\int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i\omega t) dt$ for $\omega = 0$ are not zero. This means that the signals $u(t)$ and $y(t)$ include a significant amount of the zero frequency information, which becomes bigger as the bias increases. Also, the signals must include the frequency information corresponding to the relay frequency ω_r (fundamental frequency). So, the frequency responses of the process for zero and ω_r can be estimated by (8.45) for the biased-relay test.

8.4.4 Analysis of the Fourier Analysis

Assume that the static input disturbance d_{in} is added to the process input. Note that $\int_{t_{ss}}^{t_{ss}+p_r} d_{in} \exp(-i\omega_r t) dt = 0$ for $\omega_r = 2\pi/p_r$. Then, obtain

$$\begin{aligned} \int_{t_{ss}}^{t_{ss}+p_r} (u(t) + d_{in}) \exp(-i\omega_r t) dt &= \int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i\omega_r t) dt + \int_{t_{ss}}^{t_{ss}+p_r} d_{in} \exp(-i\omega_r t) dt \\ &= \int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i\omega_r t) dt \end{aligned} \quad (8.51)$$

So, obtain the same estimate from the Fourier analysis as shown in (8.52) even in the presence of the input disturbance:

$$G(i\omega_r) = \frac{\int_{t_{ss}}^{t_{ss}+p_r} y(t) \exp(-i\omega_r t) dt}{\int_{t_{ss}}^{t_{ss}+p_r} (u(t) + d_{in}) \exp(-i\omega_r t) dt} = \frac{\int_{t_{ss}}^{t_{ss}+p_r} y(t) \exp(-i\omega_r t) dt}{\int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i\omega_r t) dt} \quad (8.52)$$

That is, the Fourier analysis of (8.45) provides the exact frequency response for the relay frequency ω_r under the circumstance of a static disturbance. Note that wrong deviation variables are equivalent to the case of static disturbances, meaning that the Fourier analysis provides an exact estimate even though wrong deviation variables are set.

Consider (8.52) for the zero frequency of $\omega = 0$:

$$\int_{t_{ss}}^{t_{ss}+p_r} (u(t) + d_{in}) \exp(-i0t) dt = \int_{t_{ss}}^{t_{ss}+p_r} u(t) dt + \int_{t_{ss}}^{t_{ss}+p_r} d_{in} dt = \int_{t_{ss}}^{t_{ss}+p_r} u(t) dt + d_{in} p_r \quad (8.53)$$

If $\int_{t_{ss}}^{t_{ss}+p_r} u(t) dt \gg d_{in} p_r$ in (8.53), then the effect of the disturbance becomes negligible, like

$$\int_{t_{ss}}^{t_{ss}+p_r} (u(t) + d_{in}) \exp(-i0t) dt \approx \int_{t_{ss}}^{t_{ss}+p_r} u(t) \exp(-i0t) dt$$

From (8.53), it is clear that the modeling error for the zero frequency can be reduced by increasing the integral $\int_{t_{ss}}^{t_{ss}+p_r} u(t) dt$. For example, the modeling error can be reduced by setting a large reference value (bias) to the biased-relay because the integral term usually becomes bigger upon increasing the reference value.

In summary, the Fourier analysis of (8.45) for the relay feedback identification has the following advantages compared with the describing function analysis. First, the frequency data set obtained is exact. Second, it can obtain several additional frequency data sets corresponding to multiples of the relay frequency and the zero frequency. Third, it provides the exact frequency data set corresponding to the relay frequency for a static disturbance. Fourth, it provides the exact frequency data set corresponding to the relay frequency even for wrongly specified deviation variables and/or an initially unsteady state. On the other hand, Fourier analysis has the disadvantage that it requires the whole data of the one period for the numerical integration. If the sampling time is not small enough, then the numerical integration becomes inaccurate. Then, Fourier analysis may result in unacceptable estimates.

Example 8.7

Activate the process $G(s) = \exp(-0.2s)/(s + 1)^2$ using a biased-relay for which the reference value is 0.5 and estimate the frequency responses using Fourier analysis.

Solution The MATLAB code for Example 8.7 is shown in Table 8.2.

8.5 Modified Fourier Transform

Fourier analysis uses only the cyclic-steady-state data points to estimate the frequency response data. So, it can provide only several frequency response data because the cyclic-steady-state data usually include a few frequency components. The modified Fourier transform is a frequency response estimator to estimate all the desired frequency data of the process (Sung and Lee, 2000). It uses all the process data from the initial transient region to the cyclic-steady-state region. The initial transient region usually includes very many frequency components. So, it is possible to estimate all the desired frequency responses of the process.

Consider the two different types of process activation in Figure 8.11 of which the initial parts are in zero steady state. Figure 8.11 is the process input and output activated by a biased-relay for which the final part is in a cyclic steady state. Figure 8.12 shows the process activation using a proportional controller for which the final part is in steady state.

The following two modified Fourier transforms can estimate the frequency responses of the process for the two cases in Figures 8.11 and 8.12.

Modified Fourier Transform for the Cyclic Steady State Assume that the process input (that is, the relay output) $u(t)$ and the process output $y(t)$ after t_{ss} are periodic with period p_r and $u(t)$ and $y(t)$ are initially in the zero steady state. Figure 8.11 is one of the examples. Then, the frequency responses of the process can be estimated by

$$G(i\omega) = \frac{[1 - \exp(-i\omega p_r)] \int_0^{t_{ss}} \exp(-i\omega\tau)y(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega\tau)y(\tau) d\tau}{[1 - \exp(-i\omega p_r)] \int_0^{t_{ss}} \exp(-i\omega\tau)u(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega\tau)u(\tau) d\tau} \quad (8.54)$$

Table 8.2 MATLAB code for the Fourier analysis in Example 8.7.

<pre> fourier_FA1.m clear; delt=0.005; tf=15; n=round(tf/delt); x=zeros(2,1); u_data=zeros(1,500); t_on=0.0; t_off=0.0; P_on=0; P_off=0; ymin=0.0; ymax=0.0; y=0.0; yref=0.5; np=0; s1=0.0; s2=0.0; s1_zero=0.0; s2_zero=0.0; index=0; y_delta=0.1; % initial phase:index=0, relay phase:index=1 for i=1:n t=i*delt; yy(i)=y; tt(i)=t; if(index==1) if(yy(i)>yref & yy(i- 1)<=yref) P_on=t-t_on; t_off=t; np=np+1; ymax_f=ymax; ymax=0.0; end if(yy(i)<=yref & yy(i- 1)>yref) P_off=t-t_off; t_on=t; ymin_f=ymin; ymin=0.0; end if(y>yref) u=-1.0; if(y>ymax) ymax=y; end end if(y<=yref) u=1.0; if(y<ymin) ymin=y; end end if(index==0) u=1.0; if(y>y_delta) index=1; end end for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u; uu(i)=u; if (np==4) % When 4th on-off, numerical integration P=P_on+P_off; w_r=2*pi/P; j=complex(0,1); s1_zero=s1_zero+y*exp(- </pre>	<pre> g_fourier_FA1.m function [next_x,y]=g_fourier_FA1(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 -1;1 -2]; B=[1;0]; C=[0 1]; delay=0.2; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return </pre> <hr/> <pre> command window >> fourier_FA1 w= 2.697, G(0)= 0.998, G(w_r)=(-0.121)+i(-0.022) </pre>
---	--

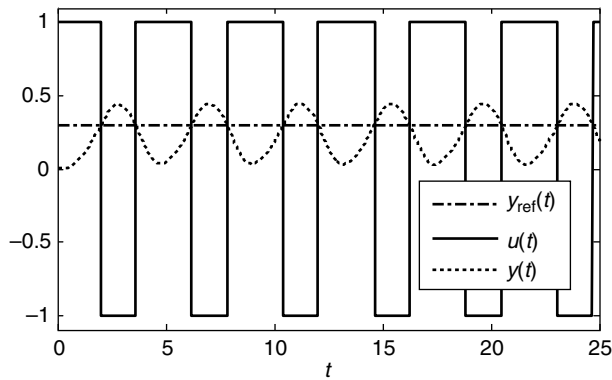
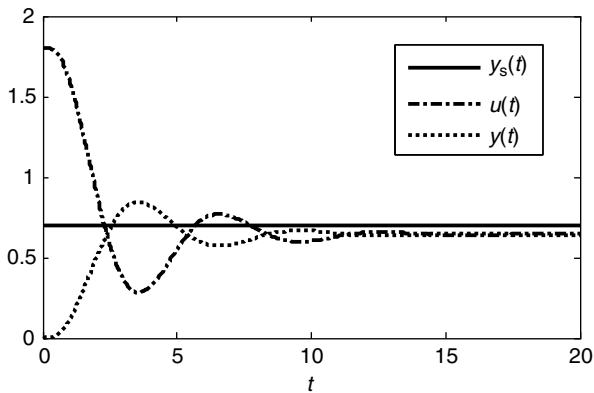
Table 8.2 (Continued)

```

j*0*t)*delt;
    s2_zero=s2_zero+u*exp(-
j*0*t)*delt;
    s1=s1+y*exp(-j*w_r*t)*delt;
    s2=s2+u*exp(-j*w_r*t)*delt;
end

[x,y]=g_fourier_FAI(x,delt,u_data);
end
fprintf('w=%6.3f, G(0)=%6.3f,
G(w_r)=(%6.3f)+i(%6.3f)
\n',w_r,s1_zero/s2_zero,
real(s1/s2),imag(s1/s2));
figure(1); plot(tt,uu,tt,yy);

```

**Figure 8.11** Process activation data for which the initial part and the final part are in a steady state and a cyclic steady state respectively.**Figure 8.12** Process activation data for which the initial part and the final part are in a steady state.

Modified Fourier Transform for the Steady State Assume that the process input $u(t)$ and the process output $y(t)$ after t_{ss} are in a steady state and $u(t)$ and $y(t)$ are initially in a zero steady state. Then, the frequency responses of the process can be estimated by

$$G(i\omega) = \frac{(i\omega)\exp(i\omega t_{ss}) \int_0^{t_{ss}} \exp(-i\omega\tau)y(\tau) d\tau + y(t_{ss})}{(i\omega)\exp(i\omega t_{ss}) \int_0^{t_{ss}} \exp(-i\omega\tau)u(\tau) d\tau + u(t_{ss})} \quad (8.55)$$

In (8.54) and (8.55), ω can be any value. So, the frequency responses of the process for all the desired frequencies can be estimated by (8.54) or (8.55) with numerical integration.

8.5.1 Derivation of the Modified Fourier Transform

The objective of the modified Fourier transform is to estimate $G(i\omega)$ for the following general linear time-invariant process (8.56) from the process input of $u(t)$ and the process output of $y(t)$:

$$G(s) = \frac{y(s)}{u(s)} = \frac{b_ms^m + b_{m-1}s^{m-1} + \dots + b_1s + b_0}{a_ns^n + a_{n-1}s^{n-1} + \dots + a_1s + 1} \quad (8.56)$$

Equation (8.56) is equivalent to (8.57) with an initially zero steady state:

$$\begin{aligned} a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \dots + a_1 \frac{dy(t)}{dt} + y(t) \\ = b_m \frac{d^m u(t)}{dt^m} + b_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \dots + b_1 \frac{du(t)}{dt} + b_0 u(t) \end{aligned} \quad (8.57)$$

Consider the transform

$$y_k(s, t) = \int_0^t \exp(-s\tau) \frac{d^k y(\tau)}{d\tau^k} d\tau, \quad k = 1, 2, \dots \quad \text{and} \quad y_0(s, t) = \int_0^t \exp(-s\tau) y(\tau) d\tau \quad (8.58)$$

The transform satisfies the property (8.59), derived by integration by parts:

$$\begin{aligned} y_n(s, t) &= \int_0^t \exp(-s\tau) \frac{d^n y(\tau)}{d\tau^n} d\tau = sy_{n-1}(s, t) + \exp(-st) \frac{d^{n-1} y(t)}{dt^{n-1}} \\ &= s^2 y_{n-2}(s, t) + s \exp(-st) \frac{d^{n-2} y(t)}{dt^{n-2}} + \exp(-st) \frac{d^{n-1} y(t)}{dt^{n-1}} \\ &= s^n y_0(s, t) + \exp(-st) \left(s^{n-2} \frac{dy(t)}{dt} + s^{n-3} \frac{d^2 y(t)}{dt^2} + \dots + \frac{d^{n-1} y(t)}{dt^{n-1}} \right) \\ &\quad + s^{n-1} \exp(-st) y(t) \\ &= s^n y_0(s, t) + \exp(-st) D_y(n-2, s, t) + s^{n-1} \exp(-st) y(t) \end{aligned} \quad (8.59)$$

where $D_y(n-2, s, t) = s^{n-2} dy(t)/dt + s^{n-3} d^2 y(t)/dt^2 + \dots + d^{n-1} y(t)/dt^{n-1}$. If apply the transform to (8.57), obtain

$$\begin{aligned} (a_ns^n + a_{n-1}s^{n-1} + \dots + a_1s + 1)y_0(s, t) + \exp(-st)(a_n D_y(n-2, s, t) + a_{n-1} D_y(n-3, s, t) \\ + \dots + a_2 D_y(0, s, t)) + \exp(-st)(a_ns^{n-1} + a_{n-1}s^{n-2} + \dots + a_1)y(t) \end{aligned}$$

$$\begin{aligned}
&= (b_m s^m + b_{m-1} s^{m-1} + \cdots + b_1 s + b_0) u_0(s, t) + \exp(-st) (b_m D_u(m-2, s, t) \\
&\quad + b_{m-1} D_u(m-3, s, t) + \cdots + b_2 D_u(0, s, t)) \\
&\quad + \exp(-st) (b_m s^{m-1} + b_{m-1} s^{m-2} + \cdots + b_1) u(t)
\end{aligned} \tag{8.60}$$

From now, let us consider the two cases. The first case is that the process input $u(t)$ and the process output $y(t)$ after t_{ss} are periodic with period p_r . The second case is that the process input $u(t)$ and the process output $y(t)$ after t_{ss} are in a steady state.

8.5.1.1 Case 1: Cyclic Steady State

Note that $d^{k-1}y(t)/dt^{k-1}|_{t_{ss}+p_r}$ and $d^{k-1}y(t)/dt^{k-1}|_{t_{ss}}$ are the same because $y(t)$ after t_{ss} is a periodic function for which the period is p_r . It is also valid for $u(t)$. So, (8.61) and (8.62) are obtained

$$\int_{t_{ss}}^{t_{ss}+p_r} \frac{d^k y(\tau)}{d\tau^k} d\tau = \frac{d^{k-1}y(t)}{dt^{k-1}} \Big|_{t_{ss}+p_r} - \frac{d^{k-1}y(t)}{dt^{k-1}} \Big|_{t_{ss}} = 0, \quad k = 1, 2, \dots \tag{8.61}$$

$$\int_{t_{ss}}^{t_{ss}+p_r} \frac{d^k u(\tau)}{d\tau^k} d\tau = \frac{d^{k-1}u(t)}{dt^{k-1}} \Big|_{t_{ss}+p_r} - \frac{d^{k-1}u(t)}{dt^{k-1}} \Big|_{t_{ss}} = 0, \quad k = 1, 2, \dots \tag{8.62}$$

And (8.63) is obtained by integrating (8.57) from t_{ss} to $t_{ss} + p_r$ and using (8.61)–(8.62):

$$\int_{t_{ss}}^{t_{ss}+p_r} y(\tau) d\tau = b_0 \int_{t_{ss}}^{t_{ss}+p_r} u(\tau) d\tau \tag{8.63}$$

Now, let us integrate (8.60) from t_{ss} to $t_{ss} + p_r$ after multiplying $\exp(st)$ and use (8.61)–(8.63) to simplify the results. Then, (8.64) is obtained

$$G(s) = \frac{b_m s^m + \cdots + b_0}{a_n s^n + \cdots + 1} = \frac{s \int_{t_{ss}}^{t_{ss}+p_r} y_0(s, t) \exp(st) dt + \int_{t_{ss}}^{t_{ss}+p_r} y(t) dt}{s \int_{t_{ss}}^{t_{ss}+p_r} u_0(s, t) \exp(st) dt + \int_{t_{ss}}^{t_{ss}+p_r} u(t) dt} \tag{8.64}$$

In (8.64), the term of $s \int_{t_{ss}}^{t_{ss}+p_r} y_0(s, t) \exp(st) dt$ can be simplified by integration by parts as follows:

$$\begin{aligned}
s \int_{t_{ss}}^{t_{ss}+p_r} y_0(s, t) \exp(st) dt &= \exp(st) \int_0^t \exp(-s\tau) y(\tau) d\tau \Big|_{t_{ss}}^{t_{ss}+p_r} - \int_{t_{ss}}^{t_{ss}+p_r} y(\tau) d\tau \\
&= \exp(st_{ss}) \left[(\exp(sp_r) - 1) \int_0^{t_{ss}} \exp(-s\tau) y(\tau) d\tau + \exp(sp_r) \right. \\
&\quad \times \left. \int_{t_{ss}}^{t_{ss}+p_r} \exp(-s\tau) y(\tau) d\tau \right] - \int_{t_{ss}}^{t_{ss}+p_r} y(\tau) d\tau
\end{aligned} \tag{8.65}$$

Then, (8.64) becomes

$$G(s) = \frac{[1 - \exp(-sp_r)] \int_0^{t_{ss}} \exp(-s\tau) y(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-s\tau) y(\tau) d\tau}{[1 - \exp(-sp_r)] \int_0^{t_{ss}} \exp(-s\tau) u(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-s\tau) u(\tau) d\tau} \tag{8.66}$$

By substituting $i\omega$ for s , the modified Fourier transform for the cyclic steady state (8.54) is obtained.

8.5.1.2 Case 2: Steady State

Note that $d^k y(t)/dt^k|_{t_{ss}} = 0$ and $d^k u(t)/dt^k|_{t_{ss}} = 0$, $k = 1, 2, \dots$, because $y(t)$ and $u(t)$ after t_{ss} are in a steady state. So, (8.57) at $t = t_{ss}$ becomes

$$y(t_{ss}) = b_0 u(t_{ss}) \quad (8.67)$$

Then, (8.68) is obtained from (8.60) at $t = t_{ss}$:

$$G(s) = \frac{s \exp(st_{ss}) \int_0^{t_{ss}} \exp(-s\tau) y(\tau) d\tau + y(t_{ss})}{s \exp(st_{ss}) \int_0^{t_{ss}} \exp(-s\tau) u(\tau) d\tau + u(t_{ss})} \quad (8.68)$$

By substituting $i\omega$ for s , the modified Fourier transform for the steady state of (8.55) is obtained.

8.5.2 Analysis of the Modified Fourier Transform

Assume that the static input disturbance d_{in} is added to the process input. Note that $\int_{t_{ss}}^{t_{ss}+p_r} d_{in} \exp(-i\omega_r \tau) d\tau = 0$ for $\omega_r = 2\pi/p_r$. Then, obtain

$$\begin{aligned} \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) (u(\tau) + d_{in}) d\tau &= \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) u(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) d_{in} d\tau \\ &= \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) u(\tau) d\tau \end{aligned} \quad (8.69)$$

Also, note that $1 - \exp(-i\omega_r p_r) = 0$ for $\omega_r = 2\pi/p_r$. So, the same estimate from the modified Fourier transform for the cyclic steady state is obtained as shown in (8.70) even in the presence of the input disturbance:

$$\begin{aligned} G(i\omega_r) &= \frac{[1 - \exp(-i\omega_r p_r)] \int_0^{t_{ss}} \exp(-i\omega_r \tau) y(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) y(\tau) d\tau}{[1 - \exp(-i\omega_r p_r)] \int_0^{t_{ss}} \exp(-i\omega_r \tau) (u(\tau) + d_{in}) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) (u(\tau) + d_{in}) d\tau} \\ &= \frac{[1 - \exp(-i\omega_r p_r)] \int_0^{t_{ss}} \exp(-i\omega_r \tau) y(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) y(\tau) d\tau}{[1 - \exp(-i\omega_r p_r)] \int_0^{t_{ss}} \exp(-i\omega_r \tau) u(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} \exp(-i\omega_r \tau) u(\tau) d\tau} \end{aligned} \quad (8.70)$$

That is, the modified Fourier transform (8.70) provides the exact frequency response for the relay frequency ω_r under the circumstance of a static disturbance. Note that wrong deviation variables are equivalent to the case of static disturbances, meaning that the modified Fourier transform provides the exact estimate even though wrong deviation variables are set.

Consider (8.71) for the zero frequency of $\omega = 0$. Also, note that $1 - \exp(-i\omega p_r) = 0$ for $\omega = 0$.

$$\int_{t_{ss}}^{t_{ss}+p_r} \exp(-i0\tau) (u(\tau) + d_{in}) d\tau = \int_{t_{ss}}^{t_{ss}+p_r} u(\tau) d\tau + \int_{t_{ss}}^{t_{ss}+p_r} d_{in} d\tau = \int_{t_{ss}}^{t_{ss}+p_r} u(\tau) d\tau + d_{in} p_r \quad (8.71)$$

From (8.71), it is clear that the modeling error for the zero frequency can be reduced by increasing the integrals $\int_{t_{ss}}^{t_{ss}+P\tau} u(\tau) d\tau$ and $\int_0^{t_{ss}} u(\tau) d\tau$. For example, the modeling error can be reduced by setting a large reference value (bias) to the biased-relay. The same conclusion can be obtained for the modified Fourier transform for the steady state.

In summary, the modified Fourier transform of (8.54) has several remarkable advantages. First, the modified Fourier transform for the cyclic steady state can provide the exact frequency response data for all the desired frequencies if the final parts of the process input and process output are in a cyclic steady state and the initial parts are in a zero steady state. Second, the modified Fourier transform for the steady state can provide the exact frequency response data for all the desired frequencies if the final parts of the process input and process output are in a steady state and the initial parts are in a zero steady state. Third, the modified Fourier transform for the cyclic steady state provides a better accuracy for the disturbance compared with the describing function analysis. Fourth, the modified Fourier transform for the cyclic steady state provides the exact frequency response data for the frequency of the relay even though wrong deviation variables are assigned.

Example 8.8

Activate the third-order plus time-delay process (8.72) using the biased-relay feedback method for the two cases of no measurement noise and measurement noise. The process output for the

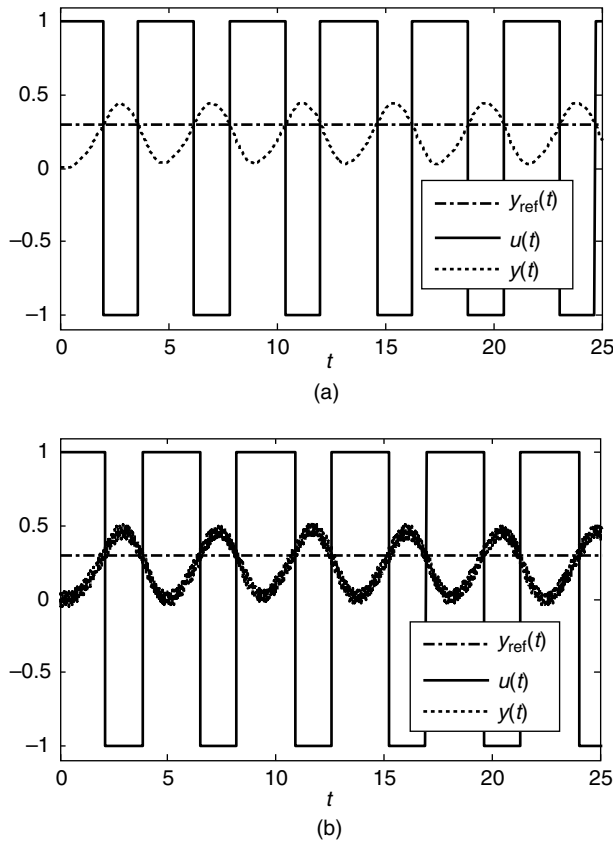


Figure 8.13 Process input and output data activated by the biased-relay feedback method in Example 8.8: (a) no measurement noise; (b) measurement noise.

measurement noise case is contaminated by random measurement noise distributed uniformly between -0.05 and 0.05 . Also, estimate the frequency responses of the process from the activated process data using the modified Fourier transform for the cyclic steady state.

$$G(s) = \frac{\exp(-0.1s)}{(s+1)^3} \quad (8.72)$$

Solution The process input and output data activated by the biased-relay feedback method is shown in Figure 8.13. The frequency response data obtained by the modified Fourier transform (8.54) are exact, as shown in Figure 8.14a. Also, the modified Fourier transform shows acceptable robustness to measurement noise, as shown in Figure 8.14b. Tables 8.3 and 8.4 show the MATLAB codes for the cases of no measurement noise and measurement noise respectively.

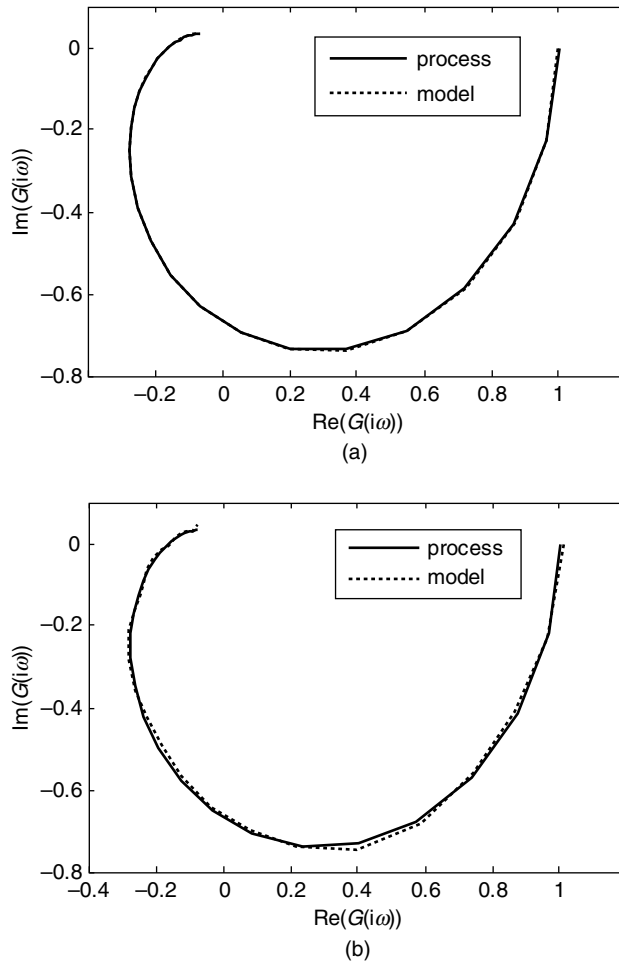


Figure 8.14 Identification results by the modified Fourier transform for the cyclic steady state in Example 8.8: (a) no measurement noise; (b) measurement noise.

Table 8.3 MATLAB code for the modified Fourier transform for the cyclic steady state in Example 8.8 with no measurement noise.

fourier_MFT1.m	g_fourier_MFT1.m
<pre> clear; delt=0.005; tf=25; n=round(tf/delt); u_data=zeros(1,500); x=zeros(3,1); t_on=0.0; t_off=0.0; P_on=0; P_off=0; y=0.0; yref=0.3; np=0; index=0; y_delta=0.1; % initial phase:index=0, relay phase:index=1 for i=1:n t=i*delt; yy(i)=y; yyref(i)=yref; tt(i)=t; if(index==1) if(yy(i)>yref & yy(i-1)<=yref) P_on=t-t_on; t_off=t; np=np+1; if np==3 tss_array=i; end if np==4 tss_pr_array=i; end end if(yy(i)<=yref & yy(i-1)>yref) P_off=t-t_off; t_on=t; end end if(y>yref) u=-1.0; end if(y<=yref) u=1.0; end if(index==0) u=1.0; if(y>y_delta) index=1; end end for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u; uu(i)=u; P=P_on+P_off; [x,y]=g_fourier_MFT1(x,delt,u_data); end for k=1:30 w(k)=(2*pi/P)*(k-1)/20; j=complex(0,1); s=j*w(k); s1=complex(0,0); s2=complex(0,0); s3=complex(0,0); s4=complex(0,0); </pre>	<pre> function [next_x,y]=g_fourier_MFT1(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 0 -1 ; 1 0 -3.0 ; 0 1 -3.0]; B=[1 ; 0 ; 0]; C=[0 0 1]; delay=0.1; delay_k=round-(delay/delt+0.00001); for i=1:n dx=A*x+B*u (500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return </pre> <hr/> <pre> command window >> fourier_MFT1 </pre>

Table 8.3 (Continued)

<pre>for i=1:tss_array-1 s1=s1+exp(- s*tt(i))*yy(i)*delt; s3=s3+exp(- s*tt(i))*uu(i)*delt; end for i=tss_array:tss_pr_array-1 s2=s2+exp(- s*tt(i))*yy(i)*delt; s4=s4+exp(- s*tt(i))*uu(i)*delt; end num_m=(1-exp(-s*P))*s1+s2; den_m=(1-exp(-s*P))*s3+s4; gjwt=num_m/den_m; Re_m(k)=real(gjwt); Im_m(k)=imag(gjwt); gjwt=exp(-0.1*s)/(s+1)^3; Re(k)=real(gjwt); Im(k)=imag(gjwt); end figure(1); plot(tt,yyref,tt,uu,tt,yy); figure(2); plot(Re,Im,'- ',Re_m,Im_m,':'); </pre>	
--	--

Table 8.4 MATLAB code for modified Fourier transform for the steady state in Example 8.8 with measurement noise.

<pre>fourier_MFT3.m clear; delt=0.005; tf=30; n=round(tf/delt); u_data=zeros(1,500); x=zeros(3,1); t_on=0.0; t_off=0.0; P_on=0; P_off=0; y=0.0; yref=0.3; np=0; index=0; y_delta=0.3; % initial phase:index=0, relay phase:index=1 hys=0.05; index_up=1; index_down=0; rand('seed',0); noise=(rand(1,n)- 0.5)*0.1; for i=1:n t=i*delt; yy(i)=y+noise(i); yyref(i)=yref; tt(i)=t; if(index==1) if(index_down==1 & index_up==0 & yy(i)<=(yref-hys) & yy(i-1)>(yref-hys)) </pre>	<pre>g_fourier_MFT3.m function [next_x,y]=g_fourier_MFT3(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 0 -1 ; 1 0 -3.0 ; 0 1 -3.0]; B=[1 ; 0 ; 0]; C=[0 0 1]; delay=0.1; delay_k=round (delay/delt+0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return command window >> fourier_MFT3 </pre>
---	--

(continued)

Table 8.4 (Continued)

```

        index_up=1; index_down=0;
        t_on=t; P_off=t_on-t_off;
    end
    if(index_up==1 &
index_down==0 & yy(i)>(yref+hys) &
yy(i-1)<=(yref+hys))
        index_up=0; index_down=1;
        t_off=t; P_on=t_off-t_on;
    np=np+1
        if np==4 tss_array=i;
    end
        if np==5 tss_pr_array=i;
    end
    end
    end
    if(index_down==1) u=-1.0; end
    if(index_up==1) u=1.0; end
    if(index==0)
        u=1.0;
        if(y>y_delta)
            index=1;
            if(yref<y_delta) u=-1.0;
        index_up=0; index_down=1; end
    end
    end
    for j=1:499
u_data(j)=u_data(j+1); end
        u_data(500)=u; uu(i)=u;
    P=P_on+P_off;

[x,y]=g_fourier_MFT3(x,delt,u_data);
end
for k=1:30
    w(k)=(2*pi/P)*(k-1)/20;
    j=complex(0,1);
    s=j*w(k); s1=0; s2=0; s3=0; s4=0;
    for i=1:tss_array-1
        s1=s1+exp(-
s*tt(i))*yy(i)*delt;
        s3=s3+exp(-
s*tt(i))*uu(i)*delt;
    end
    for i=tss_array:tss_pr_array-1
        s2=s2+exp(-
s*tt(i))*yy(i)*delt;
        s4=s4+exp(-
s*tt(i))*uu(i)*delt;
    end
    num_m=(1-exp(-s*P))*s1+s2;
    den_m=(1-exp(-s*P))*s3+s4;

```

Table 8.4 (Continued)

<pre>gjwt=num_m/den_m; Re_m(k)=real(gjwt); Im_m(k)=imag(gjwt); gjwt=exp(-0.1*s)/(s+1)^3; Re(k)=real(gjwt); Im(k)=imag(gjwt); end figure(1); plot(tt,yyref,tt,uu,tt,yy); figure(2); plot(Re,Im,'- ',Re_m,Im_m,':'); </pre>	
---	--

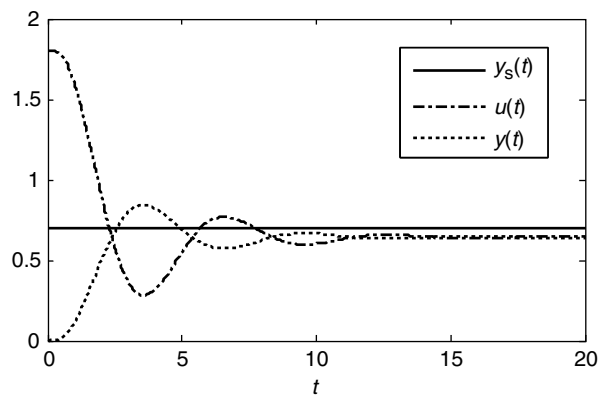


Figure 8.15 Process input and output data activated by the proportional controller in Example 8.9.

Example 8.9

Activate the third-order plus time-delay process (8.72) using a proportional controller. Also, estimate the frequency responses of the process from the activated process data using the modified Fourier transform for the steady state.

Solution The process input and output data activated by the proportional controller are shown in Figure 8.15. The frequency response data obtained by the modified Fourier transform (8.55) are exact, as shown in Figure 8.16. Table 8.5 shows the MATLAB codes.

8.6 Frequency Response Analysis with Integrals¹

Lee *et al.* (2007) proposed new process identification methods which use the integrals of the relay response instead of point data. This guarantees better accuracy and advantages in obtaining the ultimate information of the process compared with the describing function analysis

¹ Integrals of Relay Feedback Responses for Extracting Process Information, Lee *et al.* *AIChE J.* Vol. 53 Copyright ©[2007] John Wiley & Sons, Inc.

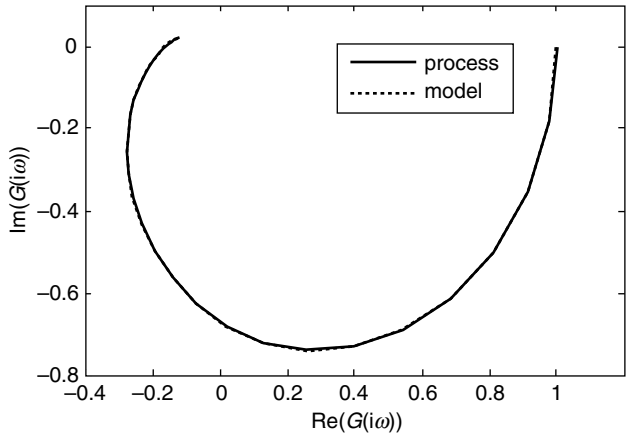


Figure 8.16 Identification results by the modified Fourier transform for the steady state in Example 8.9.

Table 8.5 MATLAB code for the modified Fourier transform for the steady state in Example 8.9.

fourier_MFT2.m	g_fourier_MFT2.m
<pre>clear; delt=0.005; tf=20; n=round(tf/delt); u_data=zeros(1,500); x=zeros(3,1); y=0.0; yref=0.7; dis=0.0; for i=1:n t=i*delt; yy(i)=y; yyref(i)=yref; tt(i)=t; u=1.8*(yref-y); for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u+dis; uu(i)=u; [x,y]=g_fourier_MFT2(x,delt,u_data); end wmax=1.2; for k=1:30 w(k)=wmax*(k-1)/20; j=complex(0,1); s=j*w(k); s1=complex(0,0); s2=complex(0,0); s3=complex(0,0); s4=complex(0,0); for i=1:n s1=s1+exp(-</pre>	<pre>function [next_x,y]=g_fourier_MFT2 (x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 0 -1 ; 1 0 -3.0 ; 0 1 -3.0]; B=[1 ; 0 ; 0]; C=[0 0 1]; delay=0.1; delay_k=round(delay/delt+ 0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return command window >> fourier_MFT2</pre>

Table 8.5 (Continued)

<pre>s*tt(i))*yy(i)*delt; s3=s3+exp(- s*tt(i))*uu(i)*delt; end num_m=s*exp(s*tt(n))*s1+yy(n); den_m=s*exp(s*tt(n))*s3+uu(n); gjwt=num_m/den_m; Re_m(k)=real(gjwt); Im_m(k)=imag(gjwt); gjwt=exp(-0.1*s)/(s+1)^3; Re(k)=real(gjwt); Im(k)=imag(gjwt); end figure(1); plot(tt,yyref,tt,uu,tt,yy); figure(2); plot(Re,Im,'-' ,Re_m,Im_m,':');</pre>	
--	--

approaches because the effects of high-harmonic terms are suppressed significantly by using the integrals of the relay responses. Because it is not required to store the process input and output and the computations are simple, it can be incorporated easily in commercial PID controllers.

8.6.1 Estimation of Ultimate Frequency Response Data

Consider the conventional relay feedback system shown in Figures 8.17 and 8.18 to derive the frequency response analysis method with integrals. The relay feedback system starts at a steady-state condition. The relay is first kept on until the process output rises to a given level and then is set to the normal mode of switching at the instant that the process output crosses a given set point. This relay feedback system will produce a stable oscillation, as shown in Figure 8.18. It is notable that the given level in the beginning of the relay feedback should be set to a significantly large value if one want to extract the zero-frequency information of the process. Otherwise, it cannot guarantee acceptable robustness in extracting the zero-frequency information from the relay responses.

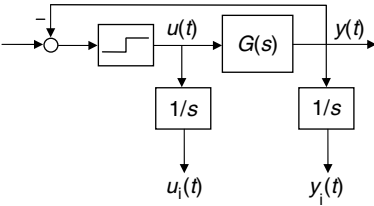


Figure 8.17 A conventional relay feedback system and the integrals of the responses. Integrals of Relay Feedback Responses for Extracting Process Information, Lee *et al.* *AIChE J.* Vol. 53 Copyright ©[2007] John Wiley & Sons, Inc.

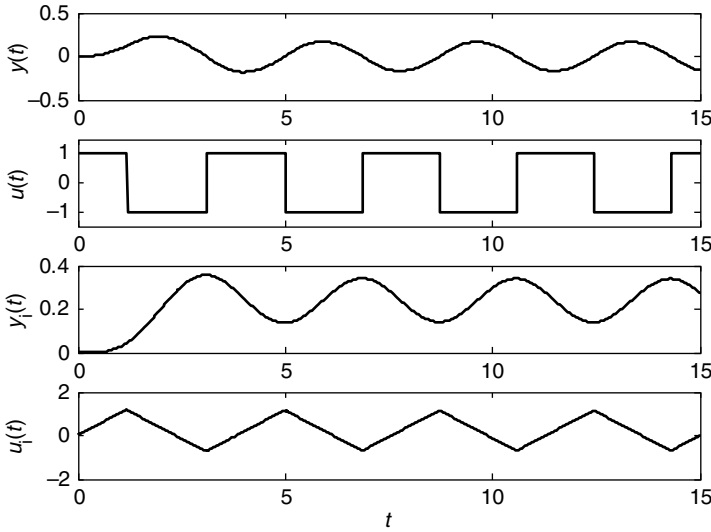


Figure 8.18 Typical relay responses and their integrals. Integrals of Relay Feedback Responses for Extracting Process Information, Lee *et al.* *AIChE J.* Vol. 53 Copyright ©[2007] John Wiley & Sons, Inc.

Let us review briefly the describing function analysis and introduce the frequency response analysis with integrals. The describing function analysis uses the oscillation data to extract approximately the ultimate frequency response of the process. Let the input and output trajectories be $u(t)$ and $y(t)$ for the conventional relay feedback system respectively. At time t_{ss} , $u(t)$ and $y(t)$ are assumed to be fully developed (cyclic steady state). This can be represented by the Fourier series as follows:

$$u(\tilde{t}) = \frac{4d}{\pi} \left[\sin(\omega\tilde{t}) + \frac{1}{3}\sin(3\omega\tilde{t}) + \frac{1}{5}\sin(5\omega\tilde{t}) + \dots \right] \quad (8.73)$$

where $\tilde{t} = t - t_{ss}$ and d is the relay magnitude. Let p_r and $\omega = 2\pi/p_r$ be the period and the frequency of the oscillation respectively. The output corresponding to $u(\tilde{t})$ is

$$y(\tilde{t}) = \frac{4d}{\pi} \left[|G(i\omega)|\sin(\omega\tilde{t} + \angle G(i\omega)) + \frac{1}{3}|G(i3\omega)|\sin(3\omega\tilde{t} + \angle G(i3\omega)) + \dots \right] \quad (8.74)$$

where $G(s)$ is the transfer function of the process. Neglecting the high-harmonic terms and assuming $\angle G(i\omega) \approx -\pi$, the ultimate frequency $\omega_u = 2\pi/p_r$, and $u(\tilde{t}) \approx 4d\sin(\omega_u\tilde{t})/\pi$, $y(\tilde{t}) \approx 4d|G(i\omega_u)|\sin(\omega_u\tilde{t} + \angle G(i\omega_u))/\pi \approx -4d|G(i\omega_u)|\sin(\omega_u\tilde{t})/\pi$ are obtained. So, the amplitude of $y(\tilde{t})$ is $4d|G(i\omega_u)|/\pi$. Then, the following approximate ultimate period of p_u and ultimate gain of k_{cu} are obtained as

$$p_u = p_r \quad (8.75)$$

$$k_{cu} = \frac{1}{|G(i\omega_u)|} = \frac{4d}{\pi a} \quad (8.76)$$

where a is the measured amplitude of $y(t)$. It should be noted that the estimated ultimate data are approximates because the high-harmonic terms are neglected. As a result, the ultimate period (8.75) and the ultimate gain (8.76) show relative errors up to 5% and 18% respectively for the FOPTD process. In this case, the ultimate gain error may not be acceptable.

The frequency response analysis method using integrals uses the integrals of the process input and output instead of the point data to obtain more accurate frequency responses of the process by suppressing the effects of the high-harmonic terms.

8.6.2 Frequency Response Estimator 1

Let $u_i(t)$ and $y_i(t)$ be the integrals of the relay responses as

$$u_i(t) = \int_0^t u(\tau) d\tau \quad (8.77)$$

$$y_i(t) = \int_0^t y(\tau) d\tau \quad (8.78)$$

From (8.73), the response $u_i(t)$ after t_{ss} is

$$u_i(\tilde{t}) = u_{im} - \frac{4d}{\pi\omega} \left[\cos(\omega\tilde{t}) + \frac{1}{9} \cos(3\omega\tilde{t}) + \frac{1}{25} \cos(5\omega\tilde{t}) + \dots \right] \quad (8.79)$$

where u_{im} is the mean value of $u_i(t)$ as follows:

$$u_{im} = \frac{1}{p_r} \int_{t_{ss}}^{t_{ss}+p_r} u_i(\tau) d\tau \quad (8.80)$$

From (8.74), the response $y_i(t)$ after t_{ss} can be represented as

$$y_i(\tilde{t}) = y_{im} - \frac{4d}{\pi\omega} \left[|G(i\omega)| \cos(\omega\tilde{t} + \angle G(i\omega)) + \frac{1}{9} |G(i3\omega)| \cos(3\omega\tilde{t} + \angle G(i3\omega)) + \dots \right] \quad (8.81)$$

where y_{im} is the mean value of $y_i(t)$:

$$y_{im} = \frac{1}{p_r} \int_{t_{ss}}^{t_{ss}+p_r} y_i(\tau) d\tau \quad (8.82)$$

Figure 8.18 shows the typical plots of these responses.

Then, $y_i(\tilde{t}) \approx y_{im} + 4d|G(i\omega_u)|\cos(\omega_u\tilde{t})/(\pi\omega_u)$ for $u_i(\tilde{t}) \approx u_{im} - 4d\cos(\omega_u\tilde{t})/(\pi\omega_u)$ is obtained by neglecting the high-harmonic terms and assuming $\angle G(i\omega) = -\pi$ (equivalently, the relay period is the ultimate period). So, the amplitude of $y_i(\tilde{t}) - y_{im}$ is $4d|G(i\omega_u)|/(\pi\omega_u)$. Then, the following ultimate gain: is obtained

$$k_{cu} = \frac{1}{|G(i\omega_u)|} = \frac{2dp_r}{\pi^2 b} \quad (8.83)$$

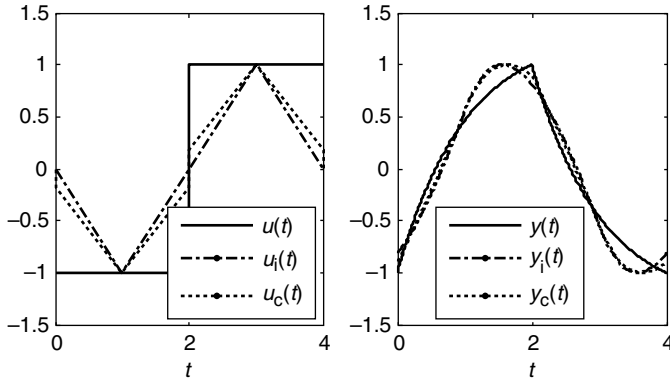


Figure 8.19 Normalized wave forms of the fully developed relay feedback responses. Integrals of Relay Feedback Responses for Extracting Process Information, Lee *et al.* *AIChE J.* Vol. 53 Copyright ©[2007] John Wiley & Sons, Inc.

where b is the amplitude of $y_i(\tilde{t}) - y_{im}$. The frequency response estimator (8.83) will be superior to (8.76) because the ratios of the high-harmonic terms to the fundamental frequency term in $y_i(\tilde{t})$ are much smaller than those in $y(\tilde{t})$, as shown in (8.73), (8.74), (8.79), and (8.81).

8.6.3 Frequency Response Estimator 2

From the relay feedback responses, the following signals of $u_c(\tilde{t})$ and $y_c(\tilde{t})$ can be constructed by combining the signals of $u(\tilde{t})$, $u_i(\tilde{t})$, and $y(\tilde{t})$, $y_i(\tilde{t})$:

$$u_c(\tilde{t}) = u(\tilde{t}) + \frac{6\pi}{p_r} [u_i(\tilde{t} + p_r/4) - u_{im}] = \frac{16d}{\pi} \sin(\omega\tilde{t}) + \frac{52d}{25\pi} \sin(5\omega\tilde{t}) + \dots \quad (8.84)$$

$$y_c(\tilde{t}) = y(\tilde{t}) + \frac{6\pi}{p_r} [y_i(\tilde{t} + p_r/4) - y_{im}] \quad (8.85)$$

Remark that $u(\tilde{t})$ is a rectangular wave, $u_i(\tilde{t})$ is a triangular wave and $u_c(\tilde{t})$ is their combination such that the third-harmonic term vanishes. The forcing functions of $u_i(\tilde{t})$ and $u_c(\tilde{t})$ are closer to a sinusoidal wave than $u(\tilde{t})$. Hence, $y_i(\tilde{t})$ and $y_c(\tilde{t})$ are closer to the sinusoidal wave than $y(\tilde{t})$ is. Figure 8.19 shows the typical plots of these responses.

Approximating the maximum of $y_c(\tilde{t})$ to

$$\max(y(\tilde{t})) + \frac{6\pi}{p_r} \max(y_i(\tilde{t}) - y_{im}) = a + \frac{6\pi b}{p_r}$$

have

$$k_{cu} = \frac{16d}{\pi(a + 6\pi b/p_r)} = \frac{1}{\frac{1}{4} \left(\frac{\pi a}{4d} \right) + \frac{3}{4} \left(\frac{\pi^2 b}{2dp_r} \right)} \quad (8.86)$$

8.6.4 Frequency Response Estimator 3

Consider the following quantity:

$$q \equiv \frac{2}{p_r} \int_{t_{ss}}^{t_{ss} + p_r} y(\tau)^2 d\tau = \frac{16d^2}{\pi^2} \left(|G(i\omega)|^2 + \frac{1}{9} |G(i3\omega)|^2 + \dots \right) \quad (8.87)$$

The right-hand side in (8.87) is derived by applying the orthogonality of sine and cosine functions to (8.74). It is remarked that, to compute the above quantity, the trajectory of $y(t)$ does not need to be stored. By ignoring the high-harmonic terms in q , (8.88) is obtained

$$k_{cu} = \frac{1}{|G(i\omega_u)|} = \frac{4d}{\pi\sqrt{q}} \quad (8.88)$$

If $y(\tilde{t})$ is sinusoidal, then $a = \sqrt{q}$ and (8.76) and (8.88) provide the same results.

8.6.5 Frequency Response Estimator 4

Consider the following quantity:

$$q_i \equiv \frac{2}{p_r} \int_{t_{ss}}^{t_{ss} + p_r} y_i(\tau)^2 d\tau - 2y_{im}^2 = \frac{16d^2}{\pi^2\omega^2} \left(|G(i\omega)|^2 + \frac{1}{81} |G(i3\omega)|^2 + \dots \right) \quad (8.89)$$

The right-hand side in (8.89) is derived by applying the orthogonality of sine and cosine functions to (8.81). By ignoring the high-harmonic terms in q_i , (8.90) is obtained

$$k_{cu} = \frac{1}{|G(i\omega_u)|} = \frac{2dp_r}{\pi^2\sqrt{q_i}} \quad (8.90)$$

If $y_i(\tilde{t})$ is sinusoidal, then $b = \sqrt{q_i}$ and (8.83) and (8.90) provide the same results.

Figure 8.20 shows the relative errors in the estimated ultimate period and ultimate gains. For the FOPTD process with the ratios of the time delay to the time constant between 0.1 and 5, (8.83), (8.86), (8.88), and (8.90) have relative errors below about 6%. This confirms that the relative errors of (8.76) for the ultimate gain can be improved considerably by the frequency response estimators using the integrals.

8.6.6 Frequency Response Estimator 5 for Nyquist Point Data

Until now the ultimate gain is estimated on the assumption that the period of relay feedback oscillation is the ultimate period. Strictly speaking, this is a wrong assumption. So, let us find the following amplitude ratio and the phase lag of the process at the frequency of the relay oscillation without the assumption:

$$G(i\omega) = A_\omega \exp[i(-\pi + \phi_\omega)], \quad \omega = 2\pi/p_r \quad (8.91)$$

where $A_\omega = |G(i\omega)|$ and $\phi_\omega = \pi + \angle G(i\omega)$. From (8.76), (8.83), (8.86), (8.88), and (8.90), the approximate amplitude ratio at the frequency $\omega = 2\pi/p_r$ can be obtained by neglecting high-harmonic terms. Among them, consider (8.90).

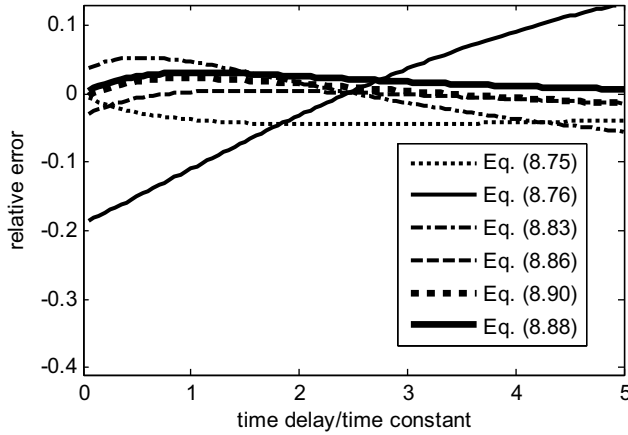


Figure 8.20 Relative error in the estimation of the ultimate period and gain for FOPTD processes $G(s) = \exp(-\theta s)/(\tau s + 1)$. Integrals of Relay Feedback Responses for Extracting Process Information, Lee *et al.* *AIChE J.* Vol. 53 Copyright ©[2007] John Wiley & Sons, Inc.

$$A_\omega = \frac{\pi^2 \sqrt{q_i}}{2dp_r} \quad (8.92)$$

Equation (8.92) is the most accurate equation for the amplitude ratio at the frequency $\omega = 2\pi/p_r$. If $|G(in\omega)| < |G(i\omega)|$, $n = 2, 3, \dots$, its approximation error is

$$|(A_\omega - |G(i\omega)|)/|G(i\omega)|| \leq \sqrt{1 + 1/3^4 + 1/5^4 + \dots} - 1 = \frac{\pi^2}{4\sqrt{6}} - 1 = 0.0073 \quad (8.93)$$

Now, consider the following quantity:

$$\begin{aligned} q_c &= \frac{\frac{2}{p_r} \int_{t_{ss}}^{t_{ss} + p_r} u(\tau) y_i(\tau) d\tau}{\frac{2}{p_r} \int_{t_{ss}}^{t_{ss} + p_r} u_i(\tau) y_i(\tau) d\tau - 2u_{im}y_{im}} \\ &= \frac{\frac{8p_r d^2}{\pi^3} \left[\sin(\angle G(i\omega)) + \frac{|G(i3\omega)|}{27|G(i\omega)|} \sin(\angle G(i3\omega)) + \dots \right]}{\frac{4p_r^2 d^2}{\pi^4} \left[\cos(\angle G(i\omega)) + \frac{|G(i3\omega)|}{81|G(i\omega)|} \cos(\angle G(i3\omega)) + \dots \right]} \\ &= \frac{2\pi}{p_r} \left\{ \frac{\sin(\phi_\omega)}{\cos(\phi_\omega)} + \frac{|G(i3\omega)|}{|G(i\omega)|} \left[-\frac{\sin(\angle G(i3\omega))}{27\cos(\phi_\omega)} + \frac{\sin(\phi_\omega)\cos(\angle G(i3\omega))}{81\cos^2(\phi_\omega)} \right] + \dots \right\} \quad (8.94) \end{aligned}$$

Ignoring the high-harmonic terms, (8.95) is obtained

$$\phi_\omega = \arctan\left(\frac{p_r}{2\pi} q_c\right) \quad (8.95)$$

Then, the approximate phase angle of the process is $\angle G(i\omega) \approx \phi\omega - \pi$. As a result, the frequency response of the process can be estimated at the frequency of the oscillation by using (8.92) and (8.95).

In summary, the frequency response estimators using the integrals of the relay responses can provide more accurate estimates for the frequency response data by reducing the high-harmonic terms. For FOPTD processes, the relative errors over 15% of the previous describing function analysis approaches in estimating the ultimate gain can be reduced to below 5%. They are very simple and can be applied easily to commercial PID controllers.

Example 8.10

Activate the process $G(s) = \exp(-0.2s)/(s + 1)^2$ and estimate the ultimate gain using (8.76), (8.83), (8.86), (8.88), and (8.90). Also, estimate the frequency response using (8.92) and (8.95).

Solution The MATLAB code for the simulation and the estimated frequency responses are given in Table 8.6.

Table 8.6 MATLAB code to estimate the ultimate gain and the frequency response using the integrals of the relay feedback signals in Example 8.10.

```

frequency_integrals1.m

clear;
delt=0.005; tf=20; n=round(tf/delt);
u_data=zeros(1,500); x=zeros(2,1);
t_on=0.0; t_off=0.0; P_on=0; P_off=0;
u=0.0; y=0.0; yref=0.0; np=0;
ui=0.0; yi=0.0; y2i=0.0; % integrals of u, y and y^2 from 0 to t
uii=0.0; yii=0.0; yi2i=0.0; % integrals of ui, yi and yi^2 from tss
to tss+pr
uyii=0.0; uiyii=0.0; % integrals of u*yi and ui*yi from tss to tss+pr
d=1.0; %magnitude of relay
ymin=10^10; ymax=-10^10; yimin=10^10; yimax=-10^10;
index=0; y_delta=0.1; % initial phase:index=0, relay phase:index=1

for i=1:n
    t=i*delt; yy(i)=y; yyref(i)=yref; tt(i)=t;
    yi=yi+y*delt;
    if(index==1)
        if(yy(i)>yref & yy(i-1)<=yref)
            P_on=t-t_on; t_off=t; np=np+1;
        end
        if(yy(i)<=yref & yy(i-1)>yref)
            P_off=t-t_off; t_on=t; end
    end

    if (np==4)
        if(ymin>y) ymin=y; end
    end
end

```

Table 8.6 (Continued)

```

    if (ymax<y) ymax=y; end
    if (yimin>yi) yimin=yi; end
    if (yimax<yi) yimax=yi; end
    yii=yii+yi*delt; y2i=y2i+y*y*delt; yi2i=yi2i+yi*yi*delt;
    uii=uii+ui*delt; uyii=uyii+u*yi*delt; uiyii=uiyii+ui*yi*delt;
end
if (y>yref) u=-d; end
if (y<=yref) u=d; end
if (index==0)
    u=d; if (y>y_delta) index=1; end
end
for j=1:499 u_data(j)=u_data(j+1); end
u_data(500)=u; uu(i)=u; P=P_on+P_off; ui=ui+u*delt;
[x,y]=g_frequency_integrals1(x,delt,u_data);
end

```

```

a_y=(ymax-ymin)/2; a_yi=(yimax-yimin)/2;
kcu_8_76=4*d/(pi*a_y); kcu_8_83=2*d*P/(pi*pi*a_yi);
kcu_8_86=16*d/(pi*(a_y+6*pi*a_yi/P));
q=2*y2i/P; kcu_8_88=4*d/pi/sqrt(q);
yim=yii/P; qi=2*yi2i/P-2*yim^2; kcu_8_90=2*d*P/(pi^2*sqrt(qi));
uim=uii/P; AR=pi^2*sqrt(qi)/(2*d*P);
qc=(2*uyii/P)/(2*uiyii/P-2*uim*yim);
PA_degree=180*(-pi+atan(P*qc/2/pi))/pi;
s=complex(0,1)*2*pi/P; G=exp(-0.2*s)/(s+1)^2;
AR_actual=abs(G); PA_actual=180*atan2(imag(G),real(G))/pi;
fprintf('kcu(real)=%6.3f\n',10.672);
fprintf('kcu(8.76)=%6.3f\n',kcu_8_76);
fprintf('kcu(8.83)=%6.3f\n',kcu_8_83);
fprintf('kcu(8.86)=%6.3f\n',kcu_8_86);
fprintf('kcu(8.88)=%6.3f\n',kcu_8_88);
fprintf('kcu(8.90)=%6.3f\n',kcu_8_90);
fprintf('AR(real)=%5.3f, PA(real)=%6.1f\n',AR_actual,PA_actual);
fprintf('AR(8.92)=%5.3f, PA(8.95)=%6.1f\n',AR,PA_degree);

```

g_frequency_integrals1.m	command window
function	>>
[next_x,y]=g_frequency_integrals1	frequency_integrals1
(x,delt,u);	kcu(real)=10.672
subdelt=delt/10; n=round(delt/subdelt);	kcu(8.76)= 9.816
A=[0 -1;1 -2]; B=[1;0]; C=[0 1]; delay=0.2;	kcu(8.83)= 9.892
delay_k=round(delay/delt);	kcu(8.86)= 9.873
for i=1:n	kcu(8.88)= 9.855
dx=A*x+B*u(500-delay_k);	kcu(8.90)= 9.862
x=x+dx*subdelt;	AR(real)=0.101,
end	PA(real)=-177.0
next_x=x; yo=C*x; y=yo;	AR(8.92)=0.101,
return	PA(8.95)=-176.4

Problems

- 8.1 Represent the periodic signals of Figure P8.1a, b and c using a Fourier series up to five terms.
- 8.2 Assume that you obtain the process output $y(t) = -a \sin(\omega t)$, $\omega = 2\pi/p$, $a > 0$ for the process input in Figure 8.1a. Find the frequency response of the process for the frequency $\omega = 2\pi/p$.
- 8.3 Assume that you obtain the process output $y(t) = b - a \sin(\omega t)$, $\omega = 2\pi/p$, $a > 0$ for the process input in Figure P8.1b. Find the frequency responses of the process for the frequencies $\omega = 0$ and $\omega = 2\pi/p$.

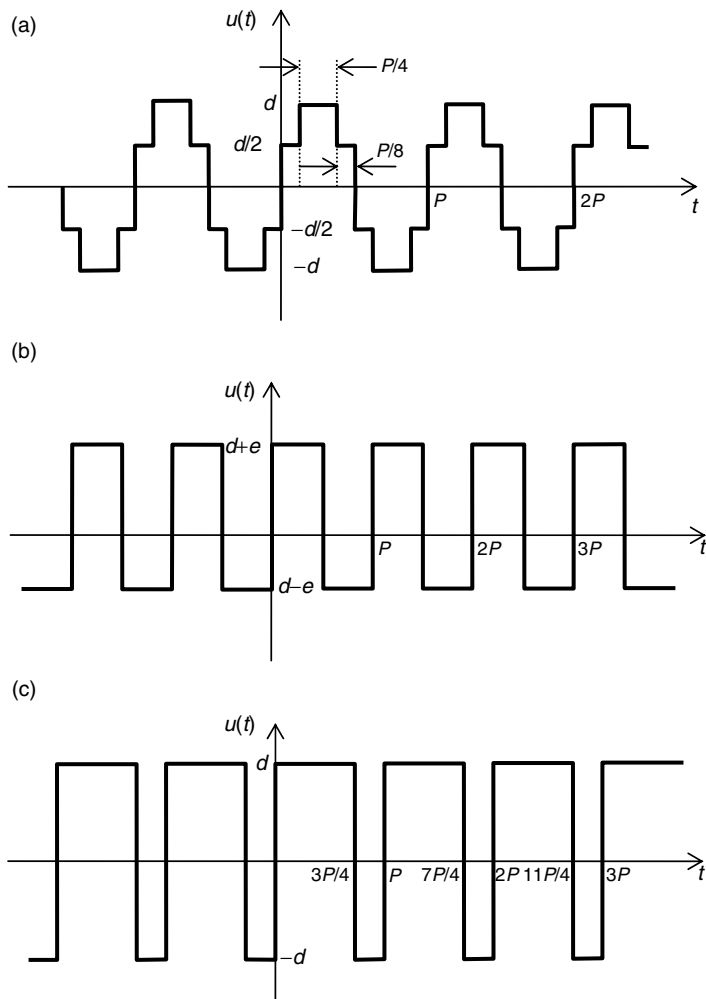


Figure 8P.1.

- 8.4 Assume that you obtain the process output $y(t) = b + a_1 \sin(\omega t - \phi_1) + a_2 \sin(2\omega t - \phi_2)$, $\omega = 2\pi/p$, $\phi_1 > 0$, $\phi_2 > 0$ for the process input in Figure P8.1c. Find the frequency responses for the frequencies $\omega = 0$, $\omega = 2\pi/p$ and $\omega = 4\pi/p$.
- 8.5 Simulate the conventional relay feedback control system for the process $G(s) = \exp(-0.1s)/(s + 1)^3$ and estimate the ultimate frequency response of the process from the activated process input and output. Also, compare the estimates with the actual ultimate frequency response.
- 8.6 Simulate the relay feedback control system combined with the time delay ($\theta = 0.3$) in Figure 8.9 for the process $G(s) = \exp(-0.1s)/(s + 1)^3$ and estimate the frequency response from the activated process input and output. Also, compare the estimates with the actual values.
- 8.7 Simulate the two-channel relay feedback control system in Figure 8.7 for the process $G(s) = \exp(-0.1s)/(s + 1)^3$ and estimate the frequency responses from the activated process inputs and outputs. Also, compare the estimates with the actual values.
- $K_p = 1.0$, $K_i = 1.0$
 - $K_p = 1.0$, $K_i = 0.0$
 - $K_p = 0.0$, $K_i = 1.0$.
- 8.8 Simulate the biased-relay feedback control system for the process $G(s) = 1/(s + 1)^4$ and estimate the two frequency responses $\omega = 0$ and $\omega = \omega_r$ from the activated process input and output using Fourier analysis. ω_r is the frequency of the relay. Also, compare them with the actual values.
- 8.9 Simulate the biased-relay feedback control system for the process $G(s) = 1/(2s + 1)(s + 1)^3$ and estimate all the frequency responses for $\omega = k\omega_r/10$, $k = 0, 1, 2, \dots, 15$, from the activated process input and output using the modified Fourier transform. Also, compare them with the actual values.
- 8.10 Simulate the proportional (P) control system of which the proportional gain is 1.0 for the process $G(s) = 1/(2s + 1)(s + 1)^3$ and estimate all the frequency responses for $\omega = k\omega_r/10$, $k = 0, 1, 2, \dots, 15$, from the activated process input and output using the modified Fourier transform. Also, compare them with the actual values.
- 8.11 Run the virtual process of Process 3 (refer to the Appendix for details) and activate the process using a biased-relay. Estimate the two frequency responses for $\omega = 0$ and $\omega = \omega_r$ from the activated process input and the output using Fourier analysis. ω_r is the frequency of the relay.
- 8.12 Estimate all the frequency responses for $\omega = k\omega_r/10$, $k = 0, 1, 2, \dots, 15$, from the activated data in Problem 8.11 using the modified Fourier transform.
- 8.13 Obtain the tuning parameters of a PID controller using the IMC tuning rule on the basis of the identified frequency responses in Problem 8.11. Also, show the control performance of the PID controller for the virtual process.
- 8.14 Obtain the tuning parameters of a PID controller using the ITAE-2 tuning rule on the basis of the identified frequency responses in Problem 8.12. Also, show the control performances of the PID controller for the virtual process.
- 8.15 Simulate the conventional relay feedback control system for the process $G(s) = \exp(-0.3s)/(s + 1)(5s + 1)$ and estimate the ultimate gains using (8.76), (8.83), (8.86), (8.88), and (8.90) and compare them with the actual value.

- 8.16 Estimate the frequency response of the relay frequency using (8.92) and (8.95) from the activated process input and output of Problem 8.15.
- 8.17 Run the virtual process of Process 1 (refer to the Appendix for details) and activate the process using a conventional relay. Estimate the ultimate gains using (8.76), (8.83), (8.86), (8.88), and (8.90).
- 8.18 Estimate the frequency response of the relay frequency using (8.92) and (8.95) from the activated process input and output of Problem 8.17.

References

- Åström, K.J. and Hägglund, T. (1984) Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, **20**, 645.
- Åström, K.J. and Hägglund, T. (1995) *PID Controllers*, Instrument Society of America, NC.
- Friman, M. and Waller, K.V. (1997) A Two-Channel Relay for Autotuning. *Industrial & Engineering Chemistry Research*, **36**, 2662.
- Kim, Y.H. (1995) PI controller tuning using modified relay feedback method. *Journal of Chemical Engineering of Japan*, **28**, 118.
- Kreyszig, E. (2006) *Advanced Engineering Mathematics*, John Wiley & Sons, Inc.
- Lee, J., Sung, S.W. and Edgar, T.F. (2007) Integrals of relay feedback responses for extracting process information. *AIChE Journal*, **53**, 2329.
- Sung, S.W. and Lee, I. (1997) Enhanced relay feedback method. *Industrial & Engineering Chemistry Research*, **36**, 5526.
- Sung, S.W. and Lee, I. (2000) An improved algorithm for automatic tuning of PID controllers. *Chemical Engineering Science*, **55**, 1883.
- Tan, K.K., Lee, T.H. and Wang, Q.G. (1996) Enhanced automatic tuning procedure for process control of PI/PID controllers. *AIChE Journal*, **42**, 2555.

9

Process Identification Methods for Continuous-Time Differential Equation Models

Two groups of process identification methods to obtain the process model in the form of continuous-time differential equation are introduced in this chapter. The first group estimates the model parameters using the least-squares method after it converts the continuous-time differential equation to the algebraic equation using the time-weighted integral transform. The second group estimates the model parameters by solving the nonlinear multivariable optimization problem of which the objective function is the norm of the modeling error. Numerical examples and MATLAB codes for each process identification method are provided.

9.1 Identification Methods Using Integral Transforms

The identification method using the integral transform provides the following continuous-time time-invariant linear model:

$$G(s) = \frac{y(s)}{u(s)} = \frac{b_ms^m + b_{m-1}s^{m-1} + \cdots + b_1s + b_0}{a_ns^n + a_{n-1}s^{n-1} + \cdots + a_1s + 1} \quad (9.1)$$

where the transfer function of $G(s)$ is strictly proper; that is, $m < n$. $u(s)$ and $y(s)$ denote the Laplace transforms of the process input (i.e. controller output) and the process output respectively. Two identification methods are introduced in this section. The first method can be applied to the case that the process is initially in a steady state. The second method can incorporate the case of the initially unsteady state.

9.1.1 Identification Method for the Case of an Initially Steady State

The identification method using the integral transform introduced in this section can be applied to the case that the process is initially in a steady state (Sung and Lee, 1999). It can estimate the

model parameters with frequency weighting. Consider the transfer function (9.1). It is equivalent to the following continuous-time differential equation (9.2). Also, assume that the initial values of the process input and the process output are zero and steady state; that is, $y(0) = 0$, $d^{i-1}y(t)/dt^{i-1}|_{t=0} = 0$, $i = 2, \dots, n$, and $u(0) = 0$, $d^{i-1}u(t)/dt^{i-1}|_{t=0} = 0$, $i = 2, \dots, m$.

$$\begin{aligned} a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \dots + a_1 \frac{dy(t)}{dt} + y(t) \\ = b_m \frac{d^m u(t)}{dt^m} + b_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \dots + b_0 u(t) + B \end{aligned} \quad (9.2)$$

where B represents a bias term to incorporate the case that the deviation variables are not specified correctly or a static disturbance enters. If the reference value for the deviation variables is chosen correctly, then by setting $B = 0$ the B value need not be estimated additionally.

Now, consider the following integral transforms for the signals of $y(t)$, $u(t)$ and $b(t) = 1$.

$$y(s) = \int_{-t_s}^{\infty} \exp(-st)y(t) dt, \quad u(s) = \int_{-t_s}^{\infty} \exp(-st)u(t) dt, \quad b(s) = \int_{-t_s}^{\infty} \exp(-st)b(t) dt \quad (9.3)$$

Because $y(0) = 0$, $d^{i-1}y(t)/dt^{i-1}|_{t=0} = 0$, $i = 2, \dots, n$, and $u(0) = 0$, $d^{i-1}u(t)/dt^{i-1}|_{t=0} = 0$, $i = 2, \dots, m$, it can be assumed that $y(t)|_{t \leq 0} = 0$, $d^{i-1}y(t)/dt^{i-1}|_{t \leq 0} = 0$, $i = 1, 2, \dots, n$, and $u(t)|_{t \leq 0} = 0$, $d^{i-1}u(t)/dt^{i-1}|_{t \leq 0} = 0$, $i = 2, \dots, m$. Then, (9.3) is equivalent to

$$y(s) = \int_0^{\infty} \exp(-st)y(t) dt, \quad u(s) = \int_0^{\infty} \exp(-st)u(t) dt, \quad b(s) = \int_{-t_s}^{\infty} \exp(-st) dt \quad (9.4)$$

It should be noted that $y(s)$ and $u(s)$ in (9.4) are the Laplace transforms. So, the following algebraic equation can be easily obtained by applying the transform to (9.2):

$$\begin{aligned} a_n s^n y(s) + a_{n-1} s^{n-1} y(s) + \dots + a_1 s y(s) + y(s) \\ = b_m s^m u(s) + b_{m-1} s^{m-1} u(s) + \dots + b_0 u(s) + B b(s) \end{aligned} \quad (9.5)$$

where $b(s) = \int_{-t_s}^{\infty} \exp(-st) dt = \int_0^{\infty} \exp[-s(t - t_s)] dt$. The objective of the process identification method is to identify the coefficients of a_k , $k = 1, 2, \dots, n$, and b_k , $k = 0, 1, \dots, m$. To do that, the process input and output data for $t \geq 0$ are required for the calculation of $y(s)$ and $u(s)$. Also, $b(s)$ can be obtained by integrating $\exp[-s(t - t_s)]$ from $t = 0$ to $t = \infty$.

Let us choose s in (9.4) like $s = \alpha + i\omega$. α is a positive real value. Then, the weight function in (9.4) becomes $\exp(-st) = \exp(-\alpha t) \exp(-i\omega t)$. Note that the magnitude $\exp(-\alpha t)$ of the weight function decays exponentially to zero as the time increases. Then, the important conclusion reached is that $y(s)$ and $u(s)$ in (9.4) can be calculated by integrating $\exp(-st)y(t)$ and $\exp(-st)u(t)$ from zero to a finite value using a numerical integration method. That is:

$$y(s) = \int_0^{t_f} \exp(-st)y(t) dt, \quad u(s) = \int_0^{t_f} \exp(-st)u(t) dt, \quad b(s) = \int_0^{t_f} \exp[-s(t - t_s)] dt \quad (9.6)$$

where α should be chosen as a positive value that makes $\exp(-\alpha t_f)$ sufficiently small. For example, $\exp(-\alpha t_f) = 0.0001 \Rightarrow \alpha = -\ln(0.0001)/t_f$. ω is for a frequency weighting. Detailed analysis about the frequency weighting will be provided in the next section.

How to determine t_s ? Consider the step input signal $u(t) = 1, t \geq 0$ and $u(t) = 0, t < 0$. If $t_s = 0$, then $u(s) = \int_0^\infty \exp(-st)u(t) dt$ is the same with $b(s) = \int_{-t_s}^\infty \exp(-st) dt$. This means that the independence between $u(s)$ and $b(s)$ cannot be guaranteed. Then, it is impossible to estimate the model parameters. So, t_s should be large enough to guarantee the independence between $u(t), t \geq 0$, and $b(t) = 1, t \geq -t_s$. The recommendation is $t_s = t_f/5$.

Now, let us recommend the procedure and specifications for the process identification method using the integral transform (9.5). First, determine α by $\alpha = -\ln(0.0001)/t_f$ and $t_s = t_f/5$, where t_f is the final time of the activated process input and the process output. Second, determine the frequencies $\omega_k, k = 1, 2, \dots, n_f$, from 0 to ω_{\max} by $\omega_k = (k-1)\omega_{\max}/(n_f-1)$, where $\omega_{\max} = 12\pi/t_f$ is recommended. Third, calculate the integrals of the transforms $y(s_k)$, $u(s_k)$ and $b(s_k)$ in (9.6) for the desired frequencies $\omega_k, k = 1, 2, \dots, n_f$. Fourth, estimate the model parameters (the coefficients of (9.2)) by applying the least-squares method to (9.5). A process order of $n = 3, m = 2$ is recommended because the third-order model can describe with acceptable accuracy the dynamics of most processes in process systems engineering.

Example 9.1

Consider the following SOPTD process:

$$G_p(s) = \frac{\exp^{-0.5s}}{(s+1)^2} \quad (9.7)$$

Figure 9.1 shows the activated process outputs by the P controller with a gain of 2. The 15 desired frequencies chosen are located equally between 0 and $12\pi/t_f$. The setpoint is changed at $t = 0$. The system identification method shows good performances, as shown in Figure 9.2. The MATLAB code to simulate Example 9.1 is shown in Tables 9.1 and 9.2. `nyquist_continuous_IT_LS0` should be executed after `continuous_IT_LS0` is executed.

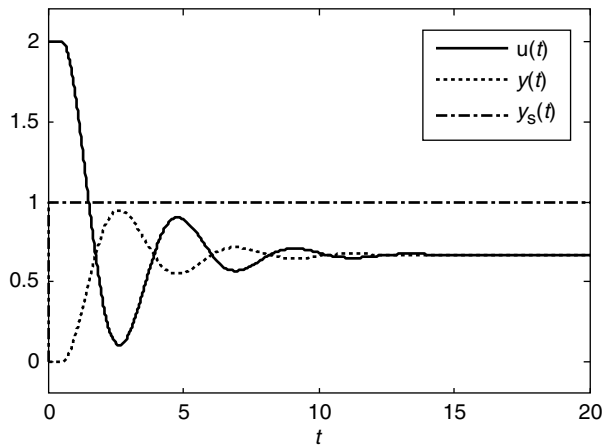


Figure 9.1 Activated process output and input by a proportional (P) controller.

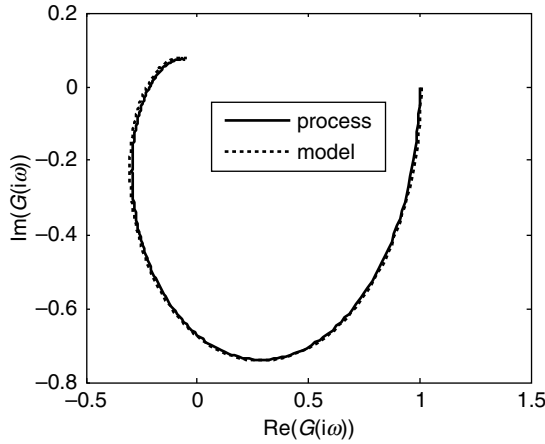


Figure 9.2 Identification results of the identification method using the integral transform.

9.1.2 Identification Method for the Case of an Initially Unsteady State

The process identification method using the integral transform in this section can manipulate the case that the process is not initially in a steady state and it can estimate the model parameters with frequency weighting. Consider the following transform for the signal $y(t)$ and $u(t)$:

$$\begin{aligned} y(n, m, \omega) &\equiv \int_0^{t_f} \frac{d^n w(\omega, t)}{dt^n} \frac{d^m y(t)}{dt^m} dt, & u(n, m, \omega) &\equiv \int_0^{t_f} \frac{d^n w(\omega, t)}{dt^n} \frac{d^m u(t)}{dt^m} dt \\ y(0, m, \omega) &\equiv \int_0^{t_f} w(\omega, t) \frac{d^m y(t)}{dt^m} dt, & u(0, m, \omega) &\equiv \int_0^{t_f} w(\omega, t) \frac{d^m u(t)}{dt^m} dt \end{aligned} \quad (9.8)$$

The weight of $w(\omega, t)$ will be explained later in this section. Using integration by parts, the following equations can be derived:

$$\begin{aligned} y(n-1, n, \omega) &= -y(n, n-1, \omega) + \frac{d^{n-1} w(\omega, t_f)}{dt^{n-1}} \frac{d^{n-1} y(t_f)}{dt^{n-1}} - \frac{d^{n-1} w(\omega, 0)}{dt^{n-1}} \frac{d^{n-1} y(0)}{dt^{n-1}} \\ u(n-1, n, \omega) &= -u(n, n-1, \omega) + \frac{d^{n-1} w(\omega, t_f)}{dt^{n-1}} \frac{d^{n-1} u(t_f)}{dt^{n-1}} - \frac{d^{n-1} w(\omega, 0)}{dt^{n-1}} \frac{d^{n-1} u(0)}{dt^{n-1}} \end{aligned} \quad (9.9)$$

Equation (9.10) can be obtained from (9.9) if the weight satisfying $d^k w(\omega, 0)/dt^k = d^k w(\omega, t_f)/dt^k = 0$, $k = 1, 2, \dots, n-1$, and $w(\omega, 0) = w(\omega, t_f) = 0$ is chosen.

$$y(0, k, \omega) = (-1)^k y(k, 0, \omega) \quad \text{and} \quad u(0, k, \omega) = (-1)^k u(k, 0, \omega) \quad k = 1, 2, \dots, n-1 \quad (9.10)$$

Now, consider the transfer function (9.1). This is equivalent to the following continuous-time differential equation:

$$a_n \frac{d^n y(t)}{dt^n} + a_{n-1} \frac{d^{n-1} y(t)}{dt^{n-1}} + \dots + a_1 \frac{dy(t)}{dt} + y(t) = b_m \frac{d^m u(t)}{dt^m} + b_{m-1} \frac{d^{m-1} u(t)}{dt^{m-1}} + \dots + b_0 u(t) + B \quad (9.11)$$

Table 9.1 MATLAB code to solve the estimation problem of Example 9.1.

<pre>continuous_IT_LS0.m clear; delt=0.02; tf=20; n=round(tf/delt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.1; x=[0 0]'; ys=0.0; y=0; u_data=zeros(1,1001); for i=1:n t=i*delt; yy(i)=y; tt(i)=t; yys(i)=ys; if (t>0.0) ys=1.0; end u=2.0*(ys-y); %u=ys; for j=1:1000 u_data(j)=u_data(j+1); end uu(i)=u; u_data(1001)=u; end [x,y]=g_continuous_IT_LS0(x,delt,u_data); end figure(1); plot(tt,uu,tt,yy,tt,yys); j=complex(0,1); nf=15; wmax=12*pi/tf; alpha=-log(0.0001)/tf; ts=-tf/5; for k=1:nf Y_0(k)=0.0; Y_1(k)=0.0; Y_2(k)=0.0; Y_3 (k)=0.0; U_0(k)=0.0; U_1(k)=0.0; U_2(k)=0.0; W_0 (k)=0.0; end for k=1:nf w(k)=(k-1)*wmax/(nf-1); s=alpha+j*w(k);</pre>	<pre>g_continuous_IT_LS0.m function [next_x,y]=g_continuous_IT_LS0(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(1000-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return command window >> continuous_IT_LS0 P_hat = 0.2570 1.4464 2.2538 0.0316 -0.3084 1.0066 -0.0058</pre>
--	--

(continued)

Table 9.1 (Continued)

<pre>for i=1:n t=i*delt; weight=exp(-s*t); weight2=exp(-s*(t-ts)); Y_0(k)=Y_0(k)+weight*yy(i)*delt; Y_1(k)=Y_1(k)+s*weight*yy(i)*delt; Y_2(k)=Y_2(k)+s^2*weight*yy(i)*delt; Y_3(k)=Y_3(k)+s^3*weight*yy(i)*delt; U_0(k)=U_0(k)+weight*uu(i)*delt; U_1(k)=U_1(k)+s*weight*uu(i)*delt; U_2(k)=U_2(k)+s^2*weight*uu(i)*delt; W_0(k)=W_0(k)+weight2*delt; end Y(k,1)=real(Y_0(k)); Y(nf+k,1)=imag(Y_0(k)); PHI(k,1)=-real(Y_3(k)); PHI(nf+k,1)=-imag(Y_3(k)); PHI(k,2)=-real(Y_2(k)); PHI(nf+k,2)=-imag(Y_2(k)); PHI(k,3)=-real(Y_1(k)); PHI(nf+k,3)=-imag(Y_1(k)); PHI(k,4)=real(U_2(k)); PHI(nf+k,4)=imag(U_2(k)); PHI(k,5)=real(U_1(k)); PHI(nf+k,5)=imag(U_1(k)); PHI(k,6)=real(U_0(k)); PHI(nf+k,6)=imag(U_0(k)); PHI(k,7)=real(W_0(k)); PHI(nf+k,7)=imag(W_0(k)); end P_hat=inv(PHI'*PHI)*(PHI'*Y)</pre>	
---	--

Table 9.2 MATLAB code to draw Figure 9.2.

```

nyquist_continuous_IT_LS0.m

del_w=0.01; wmax=pi; n=round(wmax/del_w);
for i=1:n
    w=(i-1)*del_w;
    s=j*w;
    gjw=exp(-0.5*s)/(s+1)^2;

    gjw_P_hat=(P_hat(4)*s^2+P_hat(5)*s+P_hat(6))/(P_hat(1)*s^3+
        P_hat(2)*s^2+P_hat(3)*s+1);
    Re(i)=real(gjw); Im(i)=imag(gjw);
    Re_P_hat(i)=real(gjw_P_hat); Im_P_hat(i)=imag(gjw_P_hat);
end
figure(2); plot(Re,Im,Re_P_hat,Im_P_hat,' : ');

```

command window

```

>> nyquist_continuous_IT_LS0

```

where B represents a bias term to incorporate the case that the deviation variables are not specified correctly or a static disturbance enters. If the reference value for the deviation variables is chosen correctly, then by setting $B=0$ the B value need not be estimated additionally.

The algebraic equation (9.12) can be obtained by the following two steps. Step 1, multiply $w(\omega, t)$ to (9.11) and integrate the equation from $t=0$ to $t=t_f$; Step 2, convert the algebraic equation to (9.12) using (9.10):

$$\begin{aligned}
 a_n(-1)^n y(n, 0, \omega) + a_{n-1}(-1)^{n-1} y(n-1, 0, \omega) + \cdots + a_1(-1) y(1, 0, \omega) + y(0, 0, \omega) \\
 = b_m(-1)^m u(m, 0, \omega) + b_{m-1}(-1)^{m-1} u(m-1, 0, \omega) + \cdots + b_0 u(0, 0, \omega) + Bb(0, 0, \omega)
 \end{aligned} \quad (9.12)$$

where $y(0, 0, \omega) = \int_0^{t_f} w(\omega, t) y(t) dt$, $u(0, 0, \omega) = \int_0^{t_f} w(\omega, t) u(t) dt$ and $b(0, 0, \omega) = \int_0^{t_f} w(\omega, t) dt$, and $y(k, 0, \omega) = \int_0^{t_f} (d^k w(\omega, t)/dt^k) y(t) dt$ and $u(k, 0, \omega) = \int_0^{t_f} (d^k w(\omega, t)/dt^k) u(t) dt$ for $k=1, 2, \dots, n$. Note that $b(0, 0, \omega)$, $y(k, 0, \omega)$, $k=0, 1, \dots, n$, and $u(k, 0, \omega)$, $k=0, 1, \dots, m$, in (9.12) can be calculated for various ω values for the given weight $w(\omega, t)$. Then, the coefficients a_k , $k=1, 2, \dots, n$, and b_k , $k=0, 1, 2, \dots, m$, from (9.12) can be estimated using the least-squares method.

The model of $n=3$ and $m=2$ in (9.1) can represent the dynamics of the usual processes with a good accuracy. Then, the weight should satisfy $d^k w(\omega, 0)/d\omega^k = d^k w(\omega, t_f)/d\omega^k = 0$, $k=1, 2$, and $w(\omega, 0) = w(\omega, t_f) = 0$. Let us introduce two examples for the weight.

9.1.2.1 Weight Example 1

Sung *et al.* 1998 used (9.13) and (9.14) as the weight:

$$w(\omega, t) = g(\tau, t) \exp(-i\omega t) \quad \text{with a fixed } \tau \quad (9.13)$$

$$g(\tau, t) = f(1.5\tau, t) - f(\tau, t) \quad (9.14)$$

$$f(1.5\tau, t) = \left[1 + \frac{t}{1.5\tau} + \frac{1}{2!} \left(\frac{t}{1.5\tau} \right)^2 + \frac{1}{3!} \left(\frac{t}{1.5\tau} \right)^3 + \frac{1}{4!} \left(\frac{t}{1.5\tau} \right)^4 + \frac{1}{5!} \left(\frac{t}{1.5\tau} \right)^5 \right] \exp\left(-\frac{t}{1.5\tau}\right) \quad (9.15)$$

$$f(\tau, t) = \left[1 + \frac{t}{\tau} + \frac{1}{2!} \left(\frac{t}{\tau} \right)^2 + \frac{1}{3!} \left(\frac{t}{\tau} \right)^3 + \frac{1}{4!} \left(\frac{t}{\tau} \right)^4 + \frac{1}{5!} \left(\frac{t}{\tau} \right)^5 \right] \exp\left(-\frac{t}{\tau}\right) \quad (9.16)$$

where τ and ω are related to a time weighting and a frequency weighting respectively. The Laplace transform of $g(\tau, t) = f(1.5\tau, t) - f(\tau, t)$ is $g(\tau, s) = 1/[s(\tau s + 1)^6] - 1/[s(1.5\tau s + 1)^6]$. $1/[s(\tau s + 1)^6]$ and $1/[s(1.5\tau s + 1)^6]$ are the step responses of the fast process $G(s) = 1/(\tau s + 1)^6$ and the slow process $G(s) = 1/(1.5\tau s + 1)^6$ respectively. So, $g(\tau, t) = f(1.5\tau, t) - f(\tau, t)$ satisfies the conditions of $d^k g(\tau, t)/dt^k|_{t=0} = d^k g(\tau, t)/dt^k|_{t=t_f} = 0$, $k = 1, 2, \dots, 5$ and $g(\tau, 0) = g(\tau, t_f) = 0$ if t_f/τ is big enough. $g(\tau, t) = f(1.5\tau, t) - f(\tau, t)$ determines the magnitude of the weight. So, the weight also satisfies the required conditions of $d^k w(\omega, 0)/d\omega^k = d^k w(\omega, t_f)/d\omega^k = 0$, $k = 1, 2, \dots, 5$ and $w(\omega, 0) = w(\omega, t_f) = 0$.

It is straightforward to derive the following derivatives of the weight:

$$\frac{dw(\omega, t)}{d\omega} = \frac{dg(\tau, t)}{d\omega} \exp(-i\omega t) + g(\tau, t)(-i\omega) \exp(-i\omega t) \quad (9.17)$$

$$\begin{aligned} \frac{d^2 w(\omega, t)}{d\omega^2} &= \frac{d^2 g(\tau, t)}{d\omega^2} \exp(-i\omega t) + 2 \frac{dg(\tau, t)}{d\omega} (-i\omega) \exp(-i\omega t) \\ &\quad + g(\tau, t)(-i\omega)^2 \exp(-i\omega t) \end{aligned} \quad (9.18)$$

$$\begin{aligned} \frac{d^3 w(\omega, t)}{d\omega^3} &= \frac{d^3 g(\tau, t)}{d\omega^3} \exp(-i\omega t) + 3 \frac{d^2 g(\tau, t)}{d\omega^2} (-i\omega) \exp(-i\omega t) \\ &\quad + 3 \frac{dg(\tau, t)}{d\omega} (-i\omega)^2 \exp(-i\omega t) + g(\tau, t)(-i\omega)^3 \exp(-i\omega t) \end{aligned} \quad (9.19)$$

where the derivatives of $g(\tau, t) = f(1.5\tau, t) - f(\tau, t)$ can be obtained by (9.20)–(9.23):

$$\frac{dg(\tau, t)}{d\omega} = -\frac{1}{120} \left(\frac{t}{\tau} \right)^5 \frac{\exp(-t/\tau)}{\tau} \quad (9.20)$$

$$\frac{d^2 g(\tau, t)}{d\omega^2} = \left[-\frac{1}{24} \left(\frac{t}{\tau} \right)^4 + \frac{1}{120} \left(\frac{t}{\tau} \right)^5 \right] \frac{\exp(-t/\tau)}{\tau^2} \quad (9.21)$$

$$\frac{d^3 g(\tau, t)}{d\omega^3} = \left[-\frac{1}{6} \left(\frac{t}{\tau} \right)^3 + \frac{1}{12} \left(\frac{t}{\tau} \right)^4 - \frac{1}{120} \left(\frac{t}{\tau} \right)^5 \right] \frac{\exp(-t/\tau)}{\tau^3} \quad (9.22)$$

9.1.2.2 Weight Example 2

Polynomials can be a candidate for the weight (Sung *et al.*, 2001). Consider the following weight:

$$w(\omega, t) = g(t, t_f) \exp(-i\omega t) \quad (9.23)$$

$$g(t, t_f) = \frac{t^4(t - t_f)^4}{t_f^8} \quad (9.24)$$

where t_f is the final time. It is straightforward to derive the following derivatives of the weight:

$$\frac{dw(\omega, t)}{dt} = \frac{dg(t, t_f)}{dt} \exp(-i\omega t) + g(t, t_f)(-i\omega) \exp(-i\omega t) \quad (9.25)$$

$$\frac{d^2w(\omega, t)}{dt^2} = \frac{d^2g(t, t_f)}{dt^2} \exp(-i\omega t) + 2 \frac{dg(t, t_f)}{dt} (-i\omega) \exp(-i\omega t) + g(t, t_f)(-i\omega)^2 \exp(-i\omega t) \quad (9.26)$$

$$\begin{aligned} \frac{d^3w(\omega, t)}{dt^3} &= \frac{d^3g(t, t_f)}{dt^3} \exp(-i\omega t) + 3 \frac{d^2g(t, t_f)}{dt^2} (-i\omega) \exp(-i\omega t) \\ &\quad + 3 \frac{dg(t, t_f)}{dt} (-i\omega)^2 \exp(-i\omega t) + g(t, t_f)(-i\omega)^3 \exp(-i\omega t) \end{aligned} \quad (9.27)$$

The derivatives of $g(t, t_f)$ are as follows:

$$\frac{dg(t, t_f)}{dt} = \frac{4t^3(t - t_f)^4}{t_f^8} + \frac{4t^4(t - t_f)^3}{t_f^8} \quad (9.28)$$

$$\frac{d^2g(t, t_f)}{dt^2} = \frac{12t^2(t - t_f)^4}{t_f^8} + \frac{32t^3(t - t_f)^3}{t_f^8} + \frac{12t^4(t - t_f)^2}{t_f^8} \quad (9.29)$$

$$\frac{d^3g(t, t_f)}{dt^3} = \frac{24t(t - t_f)^4}{t_f^8} + \frac{144t^2(t - t_f)^3}{t_f^8} + \frac{144t^3(t - t_f)^2}{t_f^8} + \frac{24t^4(t - t_f)}{t_f^8} \quad (9.30)$$

As shown in (9.28)–(9.30), the derivatives of $g(t, t_f)$ are zero at $t=0$ and $t=t_f$. So, the weight (9.28)–(9.30) satisfies the required conditions $d^k w(\omega, 0)/d\omega^k = d^k w(\omega, t_f)/d\omega^k = 0$, $k=1, 2, 3$ and $w(\omega, 0) = w(\omega, t_f) = 0$.

Figure 9.3 shows the magnitudes of the derivatives of the weight 1 for various ω values. Roughly speaking, only the signal between $t=0$ and the time corresponding to 30 times τ can pass through the transform. As a result, the signals outside the range from $t=0$ to $t=30\tau$ are excluded in estimating the model parameters using (9.12).

Figure 9.4 confirms that only the signal between $t=0$ and $t=t_f$ can pass through the transform. So, the signals outside the range from $t=0$ to $t=t_f$ are excluded in estimating the model parameters using (9.12).

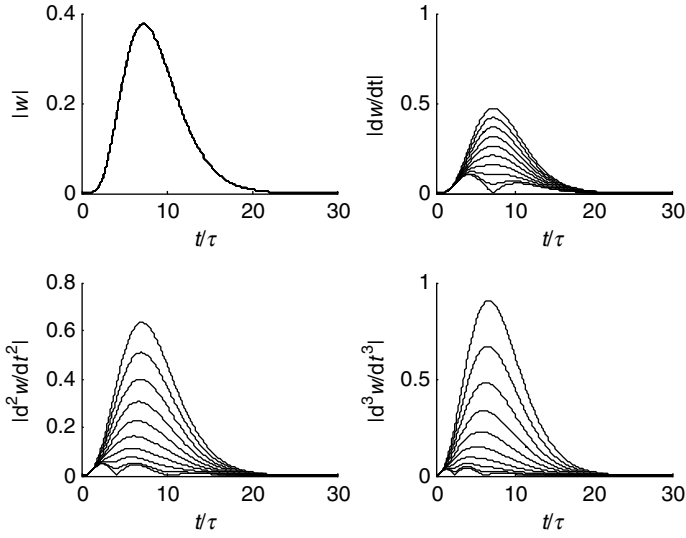


Figure 9.3 Magnitudes of the derivatives of weight 1 for various ω values.

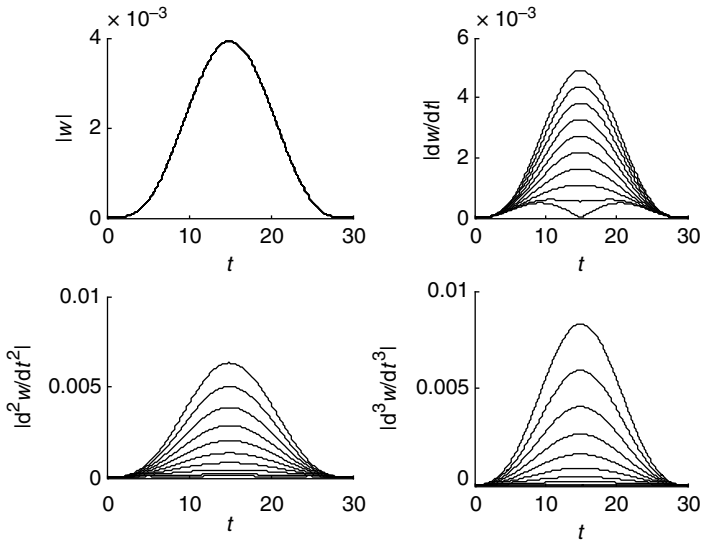


Figure 9.4 Magnitudes of the derivatives of weight 2 with $t_f = 30$ for various ω values.

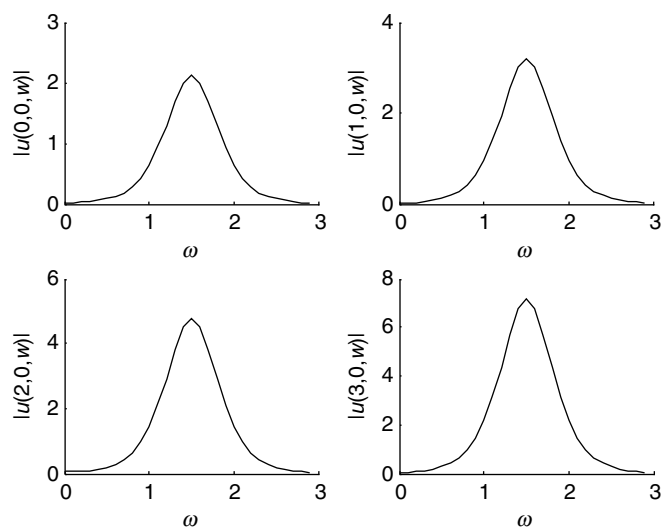


Figure 9.5 Magnitudes of the transformed signals by weight 1.

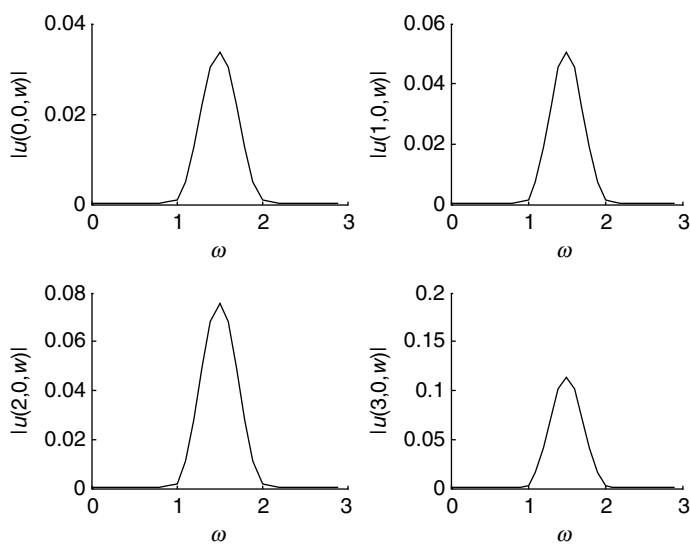


Figure 9.6 Magnitudes of the transformed signals by weight 2.

Figures 9.5 and 9.6 show the magnitudes of the transformed signals using weight 1 and weight 2 for the following two signals:

$$u_1(t) = \cos(1.5t) + \sin(1.5t) + \cos(10t) + \sin(10t), \quad u_2(t) = \cos(1.5t) + \sin(1.5t) \tag{9.31}$$

The plots for $u_1(t)$ and $u_2(t)$ are almost identical even though the high-frequency term $\cos(10t) + \sin(10t)$ is included only in $u_1(t)$, meaning that the transform amplifies the signals of the specified range of the frequency ω (in this case, from $\omega = 0$ to $\omega = 3$) and filters out the other frequency signals belonging outside the range. So, if the model parameters of the algebraic linear equation (9.12) are estimated using the least-squares method for frequencies ($\omega = \omega_k, k = 1, 2, \dots$) within the desired frequency region, then the model parameters will be adjusted to approximate mainly the desired frequency region.

The recommendation is $n = 3, m = 2$ for the model structure and $\tau = t_f/30$ for weight 1. Also, the desired frequency range from $\omega = 0$ to $\omega = 12\pi/t_f$ is recommended for weight 1 and weight 2. Note that the initial part of the setpoint change or relay on-off includes many frequency components. So, if the relay feedback method or a proportional controller is used to activate the process, then the starting time should be $t = 6\tau$ for weight 1 and $t = t_f/3$ for weight 2 to assign a big weight to the initial part.

Example 9.2

Activate the SOPTD process $G_p(s) = \exp(-0.5s)/(s + 1)^2$ using a P controller with a gain of 2. Also, obtain the process model using the integral transform with weight 1.

Solution Figure 9.7 shows the activated process output by the P controller with a gain of 2. The 10 desired frequencies chosen are located equally between $\omega = 0$ and $\omega = 12\pi/t_f$. Note that the initial process output is not in a steady state. Nevertheless, the process identification method shows a good performance, as shown in Figure 9.8. The MATLAB codes to simulate Example 9.2 are shown in Tables 9.3 and 9.4.

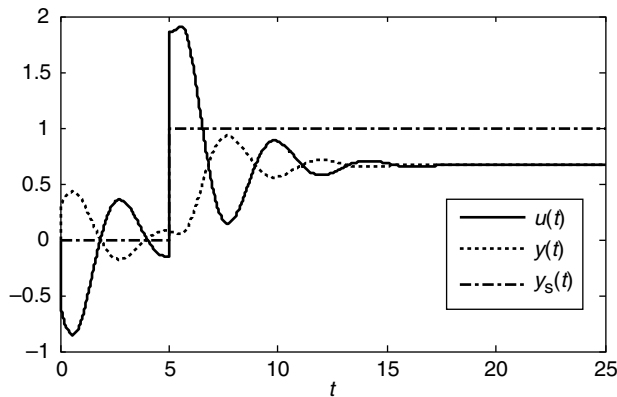


Figure 9.7 Activated process output and input by a proportional (P) controller.

Note that $y(t)$ and $u(t)$ are not independent if the setpoint is not changed, because it is a closed-loop process activation by the P controller $u(t) = k_c(y_s(t) - y(t))$. Then, the least-squares method would fail because of the singular problem originating from the dependence of $y(t)$ and $u(t)$. So, the process data must include the instance of the setpoint change to

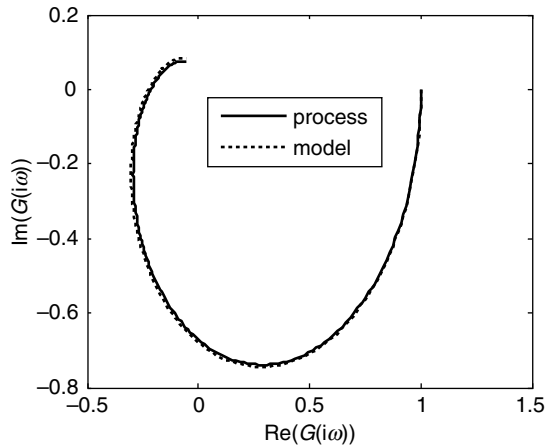


Figure 9.8 Identification results of the process identification method using the integral transform with weight 1.

guarantee the independence between $y(t)$ and $u(t)$ for a successful identification. It should also be noted that process identification is impossible for the same reason if the setpoint of the P controller changes at $t = 0$. That is, the independence between $y(t)$ and $u(t)$ by the setpoint change is eliminated because the weight function assigns a zero weighting to the initial part.

Example 9.3

Activate the SOPTD process $G_p(s) = \exp(-0.5s)/(s + 1)^2$ using a biased-relay feedback method. Also, obtain the process model using the integral transform with weight 1. The measurements of the process output are contaminated by uniformly distributed random noise between -0.05 and 0.05 . Also, a static input disturbance of 0.1 enters.

Solution Figure 9.9 shows the activated process output by the biased-relay feedback method. The process identification method using weight 1 shows an acceptable robustness for the measurement noise and the disturbance, as shown in Figure 9.10. The MATLAB codes to simulate Example 9.3 are shown in Tables 9.5 and 9.6.

Example 9.4

Activate the SOPTD process $G_p(s) = \exp(-0.5s)/(s + 1)^2$ using a P controller with a gain of 2. Also, obtain the process model using the integral transform with weight 2.

Solution Figure 9.11 shows the activated process output by a P controller with a gain of 2. The 10 desired frequencies chosen are located equally between $\omega = 0$ and $\omega = 12\pi/t_f$. The process identification method has a good performance, as shown in Figure 9.12. The MATLAB codes to simulate Example 9.4 are shown in Tables 9.7 and 9.8.

Remark If $B = 0$ is fixed, then the process identification method would show a poor performance for a static disturbance.

Table 9.3 MATLAB code to solve the estimation problem of Example 9.2.

<pre>continuous_IT_LS1.m clear; delt=0.02; tf=25; n=round(tf/delt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; pusai=[0.3; 0.5]; xi=[C; C*A]; x=inv(xi)*pusai; tau=tf/30.0; ys=0.0; y=0; u_data=zeros(1,1001); for i=1:n t=i*delt; yy(i)=y; tt(i)=t; yys(i)=ys; if t>(tau*6) ys=1.0; end u=2.0*(ys-y); for j=1:1000 u_data(j)=u_data(j+1); end uu(i)=u; u_data(1001)=u; [x,y]=g_continuous_IT_LS1(x,delt,u_data); end figure(1); plot(tt,uu,tt,yy,tt,yys); j=complex(0,1); wmax=12*pi/tf; nf=10; for k=1:nf Y_0(k)=0.0; Y_1(k)=0.0; Y_2(k)=0.0; Y_3(k)=0.0; U_0(k)=0.0; U_1(k)=0.0; U_2(k)=0.0; W_0(k)=0.0; end for k=1:nf w(k)=(k-1)*wmax/(nf-1); for i=1:n2</pre>	<pre>g_continuous_IT_LS1.m function [next_x,y]=g_continuous_IT_LS1(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(1000-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return command window >> continuous_IT_LS1 P_hat = 0.1929 1.3761 2.1867 0.0452 -0.3416 1.0013 -0.0009</pre>
--	--

```
[dnw]=dnw_dtn1(t,w(k),tau);
Y_0(k)=Y_0(k)+dnw(1)*yy(i)*delt;
Y_1(k)=Y_1(k)+dnw(2)*yy(i)*delt;
Y_2(k)=Y_2(k)+dnw(3)*yy(i)*delt;
Y_3(k)=Y_3(k)+dnw(4)*yy(i)*delt;
U_0(k)=U_0(k)+dnw(1)*uu(i)*delt;
U_1(k)=U_1(k)+dnw(2)*uu(i)*delt;
U_2(k)=U_2(k)+dnw(3)*uu(i)*delt;
W_0(k)=W_0(k)+dnw(1)*delt;
end
Y(k,1)=real(Y_0(k)); Y(nf+k,1)=imag(Y_0(k));
PHI(k,1)=real(Y_3(k)); PHI(nf+k,1)=imag(Y_3(k));
PHI(k,2)=-real(Y_2(k)); PHI(nf+k,2)=-imag(Y_2(k));
PHI(k,3)=real(Y_1(k)); PHI(nf+k,3)=imag(Y_1(k));
PHI(k,4)=real(U_2(k)); PHI(nf+k,4)=imag(U_2(k));
PHI(k,5)=-real(U_1(k)); PHI(nf+k,5)=-imag(U_1(k));
PHI(k,6)=real(U_0(k)); PHI(nf+k,6)=imag(U_0(k));
PHI(k,7)=real(W_0(k)); PHI(nf+k,7)=imag(W_0(k));
end
P_hat=inv(PHI'*PHI)*(PHI'*Y)

function dnw=dnw_dtn1(t,w,tau)
[df]=dnf_dtn1(t,tau); % d^nf(t,tau)/dt^n(t) at t
[df_tau]=dnf_dtn1(t,1.5*tau); % d^nf(t,1.5*tau)/dt^n(t) at t
g0=df_tau(1)-df(1);
g1=df_tau(2)-df(2);
g2=df_tau(3)-df(3);
g3=df_tau(4)-df(4);
g4=df_tau(5)-df(5);
j=complex(0,1);
d0w=g0*exp(-j*w*t);
```

(continued)

Table 9.3 (Continued)

<pre>d1w=g1*exp(-j*w*t)+g0*(-j*w)*exp(-j*w*t); d2w=g2*exp(-j*w*t)+2*g1*(-j*w)*exp(-j*w*t)+g0*(-j*w)^2*exp(-j*w*t); d3w=g3*exp(-j*w*t)+3*g2*(-j*w)*exp(-j*w*t)+3*g1*(-j*w)^2*exp(-j*w*t)+g0*(-j*w)^3*exp(-j*w*t); d4w=g4*exp(-j*w*t)+4*g3*(-j*w)*exp(-j*w*t)+6*g2*(-j*w)^2*exp(-j*w*t)+4*g1*(-j*w)^3*exp(-j*w*t)+g0*(-j*w)^4*exp(-j*w*t); dnw=[d0w; d1w; d2w; d3w; d4w]; % w(t), dw(t)/dt, d2w(t)/dt2, - - - return</pre>	
<pre>function df=dnf_dtn1(t,tau) d0f=(1+t/tau+(t/tau)^2/f(2)+(t/tau)^3/f(3)+(t/tau)^4/f(4)+(t/tau)^5/f(5))*exp(-t/tau); d1f=-1/120.0*(t/tau)^5*exp(-t/tau)/tau; d2f=(-1/24.0*(t/tau)^4+1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^2; d3f=(-1/6.0*(t/tau)^3+1/12.0*(t/tau)^4-1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^3; d4f=(-1/2.0*(t/tau)^2+1/2.0*(t/tau)^3-1/8.0*(t/tau)^4+1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^4; df=[d0f; d1f; d2f; d3f; d4f]; % f(t), df(t)/dt, d2f(t)/dt2, - - - return</pre>	<pre>dnf_dtn1.m</pre>
<pre>function [s]=f(n) % n! s=1.0; for i=1:n s=s*i; end return</pre>	

Table 9.4 MATLAB code to draw Figure 9.8.

<pre>nyquist_continuous_IT_LS1.m del_w=0.01; wmax=pi; n=round(wmax/del_w); for i=1:n w=(i-1)*del_w; s=j*w; gjw=exp(-0.5*s)/(s+1)^2; gjw_P_hat=(P_hat(4)*s^2+P_hat(5)*s+P_hat(6))/(P_hat(1)*s^3+P_hat(2) *s^2+P_hat(3)*s+1); Re(i)=real(gjw); Im(i)=imag(gjw); Re_P_hat(i)=real(gjw_P_hat); Im_P_hat(i)=imag(gjw_P_hat); end figure(2); plot(Re,Im,Re_P_hat,Im_P_hat,':');</pre>
<pre>command window >> nyquist_continuous_IT_LS1</pre>

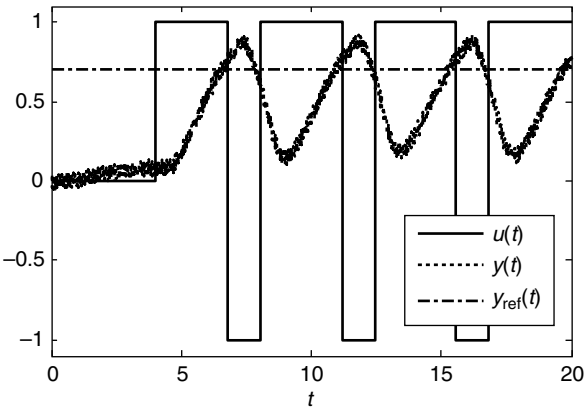


Figure 9.9 Activated process output and input by an input-biased-relay when the measured data are contaminated by the measurement noise.

9.2 Prediction Error Identification Method

Sung and Lee (2001) proposed a prediction error identification method to obtain a combined deterministic–stochastic continuous-time multi-input–single-output process model with time delays. Here, the method is simplified to the case of a deterministic continuous-time single-input–single-output process model with time delay. It identifies the model parameters by minimizing the prediction error using the Levenberg–Marquardt optimization method with the exact derivatives of the objective function with respect to the adjustable parameters. Compared

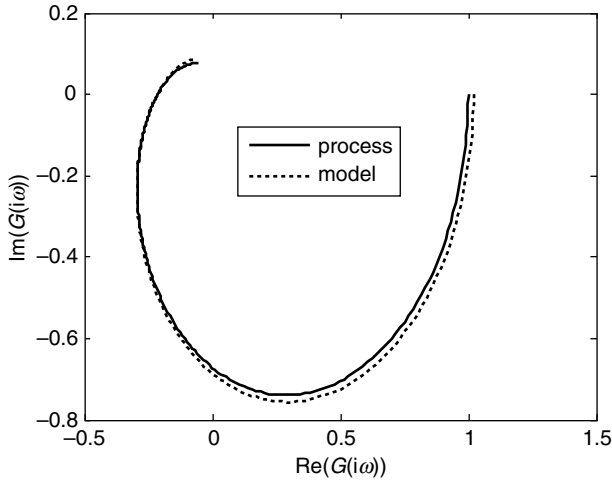


Figure 9.10 Identification results of the process identification method using the integral transform with weight 1 in Example 9.3.

with discrete-time identification methods, it does not suffer from the small sampling time or irregular sampling problem. Also, it can be applied to a large sampling time that cannot be incorporated by the previous continuous-time approaches using transforms. It can also identify the time delay term and the other model parameters simultaneously in a systematic way.

9.2.1 State-Space Process

Consider the following state-space representation:

$$\frac{dx(t)}{dt} = Ax(t) + Bu(t - \theta) \quad (9.32)$$

$$y(t) = Cx(t) \quad (9.33)$$

where $u(t - \theta)$ and $y(t)$ denote the delayed process input and the scalar process output respectively. $x(t)$ is the n -dimensional state. The objective of this method is to estimate the matrices A , B and θ from the discrete-sampled-output data and the given process input.

$$A = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & -a_n \\ 1 & 0 & 0 & \cdots & 0 & -a_{n-1} \\ 0 & 1 & 0 & \cdots & 0 & -a_{n-2} \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ 0 & 0 & 0 & \cdots & 0 & -a_2 \\ 0 & 0 & 0 & \cdots & 1 & -a_1 \end{bmatrix} \quad (9.34)$$

(continued)

Table 9.5 MATLAB code to solve the estimation problem of Example 9.3.

<pre>continuous_IT_LS2.m clear; delt=0.01; tf=20; n=round(tf/delt); tau=tf/30.0; u_data=zeros(1,1000); x=zeros(2,1); t_on=0.0; t_off=0.0; P_on=0; P_off=0; y=0.0; yref=0.7; np=0; dis=0.1; index=0; y_delta=0.2; % initial phase:index=0, relay phase:index=1 hys=0.1; index_up=1; index_down=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.1; u=0.0; for i=1:n t=i*delt; yy(i)=y+noise(i); yyref(i)=yref; tt(i)=t; if (index==0 & t>(6*tau)) u=1.0; if (y>y_delta) index=1; end; end if (index==1) if (index_down==1 & index_up==0 & yy(i) <= (yref-hys) & yy(i-1) > (yref-hys)) index_up=1; index_down=0; t_on=t; P_off=t_on-t_off; end if (index_up==1 & index_down==0 & yy(i) > (yref+hys) & yy(i-1) <= (yref+hys)) index_up=0; index_down=1; t_off=t; P_on=t_off-t_on; np=np+1; end end if (index==1) if (index_down==1) u=-1.0; end if (index_up==1) u=1.0; end end for j=1:999 u_data(j)=u_data(j+1); end u_data(1000)=u+dis; uu(i)=u; P=P_on+P_off;</pre>	<pre>g_continuous_IT_LS2.m function [next_x,y]=g_continuous_IT_LS2(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(1000-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return</pre>
	<pre>command window >> continuous_IT_LS2 P_hat = 0.1386 1.3444 2.1625 0.0420 -0.3482 1.0099 0.0953</pre>

Table 9.5 (Continued)

```

[x,y]=g_continuous_IT_Is2(x,delt,u_data);
end
figure(1); plot(tt,uu,tt,yy,tt,yyref);

j=complex(0,1); wmax=12*pi/tf; nf=10;
for k=1:nf
    Y_0(k)=0.0; Y_1(k)=0.0; Y_2(k)=0.0; Y_3(k)=0.0;
    U_0(k)=0.0; U_1(k)=0.0; U_2(k)=0.0; W_0(k)=0.0;
end
for k=1:nf
    w(k)=(k-1)*wmax/(nf-1);
    for i=1:n
        t=i*delt;
        [dnw]=dnw_dtn2(t,w(k),tau);
        Y_0(k)=Y_0(k)+dnw(1)*yy(i)*delt;
        Y_1(k)=Y_1(k)+dnw(2)*yy(i)*delt;
        Y_2(k)=Y_2(k)+dnw(3)*yy(i)*delt;
        Y_3(k)=Y_3(k)+dnw(4)*yy(i)*delt;
        U_0(k)=U_0(k)+dnw(1)*uu(i)*delt;
        U_1(k)=U_1(k)+dnw(2)*uu(i)*delt;
        U_2(k)=U_2(k)+dnw(3)*uu(i)*delt;
        W_0(k)=W_0(k)+dnw(1)*delt;
    end
    Y(k,1)=real(Y_0(k)); Y(nf+k,1)=imag(Y_0(k));
    PHI(k,1)=real(Y_3(k)); PHI(nf+k,1)=imag(Y_3(k));
    PHI(k,2)=-real(Y_2(k)); PHI(nf+k,2)=-imag(Y_2(k));
    PHI(k,3)=real(Y_1(k)); PHI(nf+k,3)=imag(Y_1(k));
    PHI(k,4)=real(U_2(k)); PHI(nf+k,4)=imag(U_2(k));
    PHI(k,5)=-real(U_1(k)); PHI(nf+k,5)=-imag(U_1(k));
    PHI(k,6)=real(U_0(k)); PHI(nf+k,6)=imag(U_0(k));
    PHI(k,7)=real(W_0(k)); PHI(nf+k,7)=imag(W_0(k));
end
P_hat=inv(PHI'*PHI)*(PHI'*Y)

```



```

dnw_dtn2.m

function dnw=dnw_dtn2(t,w,tau)
[df]=dnf_dtn2(t,tau); % d^n f(t,tau)/dt^n(t) at t
[df_tau]=dnf_dtn2(t,1.5*tau); % d^n f(t,1.5*tau)/dt^n(t) at t
g0=df_tau(1)-df(1);
g1=df_tau(2)-df(2);
g2=df_tau(3)-df(3);
g3=df_tau(4)-df(4);
g4=df_tau(5)-df(5);
j=complex(0,1);
d0w=g0*exp(-j*w*t);
d1w=g1*exp(-j*w*t)+g0*(-j*w)*exp(-j*w*t);
d2w=g2*exp(-j*w*t)+2*g1*(-j*w)*exp(-j*w*t)+g0*(-j*w)^2*exp(-j*w*t);
d3w=g3*exp(-j*w*t)+3*g2*(-j*w)*exp(-j*w*t)+3*g1*(-j*w)^2*exp(-j*w*t)+g0*(-j*w)^3*exp(-j*w*t);
d4w=g4*exp(-j*w*t)+4*g3*(-j*w)*exp(-j*w*t)+6*g2*(-j*w)^2*exp(-j*w*t)+4*g1*(-j*w)^3*exp(-j*w*t)+g0*(-j*w)^4*exp(-j*w*t);
dnw=[d0w; d1w; d2w; d3w; d4w]; % w(t), dw(t)/dt, d2w(t)/dt2, - - -
return

```

```

dnf_dtn2.m

function df=dnf_dtn2(t,tau)
d0f=(1+t/tau+(t/tau)^2/f(2)+(t/tau)^3/f(3)+(t/tau)^4/f(4)+(t/tau)^5/f(5))*exp(-t/tau);
d1f=-1/120.0*(t/tau)^5*exp(-t/tau)/tau;
d2f=(-1/24.0*(t/tau)^4+1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^2;
d3f=(-1/6.0*(t/tau)^3+1/12.0*(t/tau)^4-1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^3;
d4f=(-1/2.0*(t/tau)^2+1/2.0*(t/tau)^3-1/8.0*(t/tau)^4+1/120.0*(t/tau)^5)*exp(-t/tau)/(tau)^4;
df=[d0f; d1f; d2f; d3f; d4f]; % f(t), df(t)/dt, d2f(t)/dt2, - - -
return

function [s]=f(n) % n!
s=1.0;
for i=1:n
    s=s*i;
end
return

```

Table 9.6 MATLAB code to draw Figure 9.8.

```

nyquist_continuous_IT_LS2.m

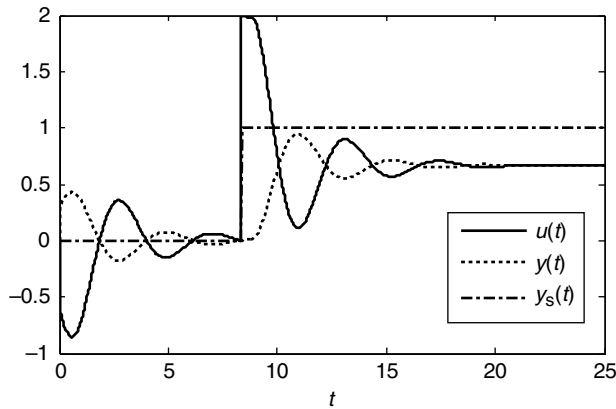
del_w=0.01; wmax=pi; n=round(wmax/del_w);
for i=1:n
    w=(i-1)*del_w;
    s=j*w;
    gjw=exp(-0.5*s)/(s+1)^2;

    gjw_P_hat=(P_hat(4)*s^2+P_hat(5)*s+P_hat(6))/(P_hat(1)*s^3+P_hat(2)
    *s^2+P_hat(3)*s+1);
    Re(i)=real(gjw); Im(i)=imag(gjw);
    Re_P_hat(i)=real(gjw_P_hat); Im_P_hat(i)=imag(gjw_P_hat);
end
figure(2); plot(Re,Im,Re_P_hat,Im_P_hat,':');

```

command window

```
>> nyquist_continuous_IT_LS2
```

**Figure 9.11** Activated process output and input by a proportional (P) controller.

$$B = \begin{bmatrix} b_n & b_{n-1} & b_{n-2} & \cdots & b_2 & b_1 \end{bmatrix}^T \quad (9.35)$$

$$C = \begin{bmatrix} 0 & 0 & 0 & \cdots & 0 & 1 \end{bmatrix} \quad (9.36)$$

The state-space process (9.32)–(9.36) is equivalent to the following transfer function if the process is initially in a zero steady state:

$$G(s) = \frac{y(s)}{u(s)} = \frac{b_1 s^{n-1} + b_2 s^{n-2} + \cdots + b_{n-1} s + b_n}{s^n + a_1 s^{n-1} + \cdots + a_{n-1} s + a_n} \exp(-\theta s) \quad (9.37)$$

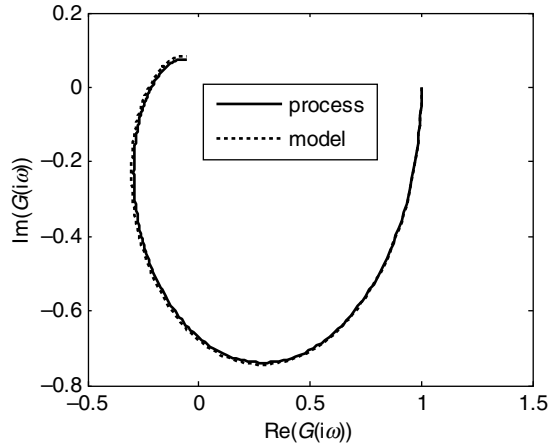


Figure 9.12 Identification results of the process identification method using the integral transform with weight 2.

For more details on the state-space model, refer to Chapter 1.

9.2.2 Continuous-Time Prediction Error Identification Method

The prediction error identification method in this section estimates the model parameters by solving the following nonlinear optimization problem:

$$\min_{\hat{A}, \hat{B}, \hat{\theta}} \left[V(\hat{A}, \hat{B}, \hat{\theta}) = \frac{0.5}{N} \sum_{i=1}^N (y(t_i) - \hat{y}(t_i))^2 \right] \quad (9.38)$$

subject to

$$\frac{d\hat{x}(t)}{dt} = \hat{A}\hat{x}(t) + \hat{B}u(t - \hat{\theta}) \quad (9.39)$$

$$\hat{y}(t) = C\hat{x}(t) \quad (9.40)$$

$$t_0 = 0 < t_1 < \cdots < t_{N-1} < t_N \quad (9.41)$$

where $y(t)$ and $\hat{y}(t)$ denote the process output and the model output respectively. To solve this optimization problem, the Levenberg–Marquardt method is used, which repeats (9.42) until the parameters converge.

$$\hat{p}(j) = \hat{p}(j-1) - \left[\frac{\partial^2 V(\hat{p}(j-1))}{\partial \hat{p}^2} + \alpha I \right]^{-1} \left[\frac{\partial V(\hat{p}(j-1))}{\partial \hat{p}} \right] \quad (9.42)$$

$$\hat{p} = [\hat{a}_1 \quad \hat{a}_2 \quad \cdots \quad \hat{a}_n \quad \hat{b}_1 \quad \hat{b}_2 \quad \cdots \quad \hat{b}_n \quad \hat{\theta}]^T \quad (9.43)$$

Table 9.7 MATLAB code to solve the estimation problem of Example 9.4.

<pre>continuous_IT_LS3.m clear; delt=0.02; tf=25; n=round(tf/delt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; pusai=[0.3; 0.5]; xi=[C; C*A]; x=inv(xi)*pusai; ys=0.0; y=0; u_data=zeros(1,1001); for i=1:n t=i*delt; yy(i)=y; tt(i)=t; yys(i)=ys; if t>(tf/3.0) ys=1.0; end u=2.0*(ys-y); for j=1:1000 u_data(j)=u_data(j+1); end uu(i)=u; u_data(1001)=u; [x,y]=g_continuous_IT_LS3(x,delt,u_data); end figure(1); plot(tt,uu,tt,yy,tt,yys); j=complex(0,1); wmax=12*pi/tf; nf=10; for k=1:nf Y_0(k)=0.0; Y_1(k)=0.0; Y_2(k)=0.0; Y_3(k)=0.0; U_0(k)=0.0; U_1(k)=0.0; U_2(k)=0.0; W_0(k)=0.0; end for k=1:nf w(k)=(k-1)*wmax/(nf-1); for i=1:n t=i*delt;</pre>	<pre>g_continuous_IT_LS3.m function [next_x,y]=g_continuous_IT_LS3(x,delt,u); subdelt=delt/10; n=round(delt/subdelt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.5; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(1000-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return command window >> continuous_IT_LS3 P_hat = 0.1924 1.3753 2.1863 0.0449 -0.3420 1.0008 -0.0005</pre>
---	--

```

[dnw]=dnw_dtn3(t,w(k),tf);
Y_0(k)=Y_0(k)+dnw(1)*yy(i)*delt;
Y_1(k)=Y_1(k)+dnw(2)*yy(i)*delt;
Y_2(k)=Y_2(k)+dnw(3)*yy(i)*delt;
Y_3(k)=Y_3(k)+dnw(4)*yy(i)*delt;
U_0(k)=U_0(k)+dnw(1)*uu(i)*delt;
U_1(k)=U_1(k)+dnw(2)*uu(i)*delt;
U_2(k)=U_2(k)+dnw(3)*uu(i)*delt;
W_0(k)=W_0(k)+dnw(1)*delt;
end
Y(k,1)=real(Y_0(k)); Y(nf+k,1)=imag(Y_0(k));
PHI(k,1)=real(Y_3(k)); PHI(nf+k,1)=imag(Y_3(k));
PHI(k,2)=-real(Y_2(k)); PHI(nf+k,2)=-imag(Y_2(k));
PHI(k,3)=real(Y_1(k)); PHI(nf+k,3)=imag(Y_1(k));
PHI(k,4)=real(U_2(k)); PHI(nf+k,4)=imag(U_2(k));
PHI(k,5)=-real(U_1(k)); PHI(nf+k,5)=-imag(U_1(k));
PHI(k,6)=real(U_0(k)); PHI(nf+k,6)=imag(U_0(k));
PHI(k,7)=real(W_0(k)); PHI(nf+k,7)=imag(W_0(k));
end
P_hat=inv(PHI'*PHI)*(PHI'*Y)

```

```

dnw_dtn3.m

function dnw=dnw_dtn3(t,w,tf)
[df]=dnf_dtn3(t,tf); % d^nf(t,tau)/dt^n(t) at t
j=complex(0,1);
d0w=df(1)*exp(-j*w*t);
d1w=df(2)*exp(-j*w*t)+df(1)*(-j*w)*exp(-j*w*t);
d2w=df(3)*exp(-j*w*t)+2*df(2)*(-j*w)*exp(-j*w*t)+df(1)*(-j*w)^2*exp(-j*w*t);
d3w=df(4)*exp(-j*w*t)+3*df(3)*(-j*w)*exp(-j*w*t)+3*df(2)*(-j*w)^2*exp(-j*w*t)+df(1)*(-j*w)^3*exp(-j*w*t);
dnw=[d0w; d1w; d2w; d3w]; % w(t), dw(t)/dt, d2w(t)/dt2, ---
return

```

(continued)

Table 9.7 (Continued)

	dnf_dtn3.m
<pre>function df=dnf_dtn3(t,tf) d0f=t^4*(t-tf)^4/tf^8; d1f=4*t^3*(t-tf)^4/tf^8+4*t^4*(t-tf)^3/tf^8; d2f=12*t^2*(t-tf)^4/tf^8+32*t^3*(t-tf)^3/tf^8+12*t^4*(t-tf)^2/tf^8; d3f=24*t*(t-tf)^4/tf^8+144*t^2*(t-tf)^3/tf^8+144*t^3*(t-tf)^2/tf^8+24*t^4*(t-tf)/tf^8; df=[d0f; d1f; d2f; d3f]; % f(t), df(t)/dt, d2f(t)/dt2, - - - return</pre>	

Table 9.8 MATLAB code to draw Figure 9.12.

```
nyquist_continuous_IT_LS3.m
```

```
del_w=0.01; wmax=pi; n=round(wmax/del_w);
for i=1:n
    w=(i-1)*del_w;
    s=j*w;
    gjw=exp(-0.5*s)/(s+1)^2;

    gjw_P_hat=(P_hat(4)*s^2+P_hat(5)*s+P_hat(6))/(P_hat(1)*s^3+P_hat(2)
    *s^2+P_hat(3)*s+1);
    Re(i)=real(gjw); Im(i)=imag(gjw);
    Re_P_hat(i)=real(gjw_P_hat); Im_P_hat(i)=imag(gjw_P_hat);
end
figure(2); plot(Re,Im,Re_P_hat,Im_P_hat,':');
```

```
command window
```

```
>> nyquist_continuous_IT_LS3
```

where j denotes the iteration number and α is a small positive value that can be updated every iteration to compromise between the robustness and the convergence rate. For details, refer to Chapter 2.

The initial values for the Levenberg–Marquardt optimization method can be determined roughly by previous approaches. For example, the continuous-time (or discrete-time) high-order autoregressive exogenous input (ARX) model using the process identification methods of Section 9.1 (or Section 10.2) is obtained and the high-order model can be reduced to a continuous-time low-order plus time-delay model using the model reduction (or conversion) methods of Section 5.8 (or Chapter 11). Then, the low-order plus time-delay model obtained can be used as the initial parameters for the Levenberg–Marquardt optimization method.

The partial derivatives of the objective function with respect to the adjustable parameters in (9.42) can be calculated by the numerical derivative or solving the differential equations. Refer to Chapter 2 for the numerical derivative. Consider the following to understand how to calculate the partial derivatives by solving the differential equations.

From (9.38), (9.44) is derived:

$$\frac{\partial V(\hat{p})}{\partial \hat{p}} = -\frac{1}{N} \sum_{i=1}^N (y(t_i) - \hat{y}(t_i)) \frac{\partial \hat{y}(t_i)}{\partial \hat{p}} \quad (9.44)$$

$$\frac{\partial \hat{y}(t)}{\partial \hat{p}} = \begin{bmatrix} \frac{\partial \hat{y}(t)}{\partial \hat{a}_1} & \cdots & \frac{\partial \hat{y}(t)}{\partial \hat{a}_n} & \frac{\partial \hat{y}(t)}{\partial \hat{b}_1} & \cdots & \frac{\partial \hat{y}(t)}{\partial \hat{b}_n} & \frac{\partial \hat{y}(t)}{\partial \hat{\theta}} \end{bmatrix}^T \quad (9.45)$$

$$\frac{\partial^2 V(\hat{p})}{\partial \hat{p}^2} = -\frac{1}{N} \sum_{i=1}^N (y(t_i) - \hat{y}(t_i)) \frac{\partial^2 \hat{y}(t_i)}{\partial \hat{p}^2} + \frac{1}{N} \sum_{i=1}^N \begin{bmatrix} \frac{\partial \hat{y}(t_i)}{\partial \hat{p}} \end{bmatrix} \begin{bmatrix} \frac{\partial \hat{y}(t_i)}{\partial \hat{p}} \end{bmatrix}^T \quad (9.46)$$

$$\frac{\partial^2 \hat{y}(t)}{\partial \hat{p}^2} = \begin{bmatrix} \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_1 \partial \hat{a}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_1 \partial \hat{a}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_1 \partial \hat{b}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_1 \partial \hat{b}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_1 \partial \hat{\theta}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_n \partial \hat{a}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_n \partial \hat{a}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_n \partial \hat{b}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_n \partial \hat{b}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{a}_n \partial \hat{\theta}} \\ \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_1 \partial \hat{a}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_1 \partial \hat{a}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_1 \partial \hat{b}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_1 \partial \hat{b}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_1 \partial \hat{\theta}} \\ \vdots & \ddots & \vdots & \vdots & \ddots & \vdots & \vdots \\ \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_n \partial \hat{a}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_n \partial \hat{a}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_n \partial \hat{b}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_n \partial \hat{b}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{b}_n \partial \hat{\theta}} \\ \frac{\partial^2 \hat{y}(t)}{\partial \hat{\theta} \partial \hat{a}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{\theta} \partial \hat{a}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{\theta} \partial \hat{b}_1} & \dots & \frac{\partial^2 \hat{y}(t)}{\partial \hat{\theta} \partial \hat{b}_n} & \frac{\partial^2 \hat{y}(t)}{\partial \hat{\theta} \partial \hat{\theta}} \end{bmatrix} \quad (9.47)$$

From (9.39) and (9.40), (9.48)–(9.58) are obtained:

$$\frac{\partial \hat{y}(t)}{\partial \hat{p}} = C \frac{\partial \hat{x}(t)}{\partial \hat{p}} \quad \frac{\partial^2 \hat{y}(t)}{\partial \hat{p}_i \partial \hat{p}_j} = C \frac{\partial^2 \hat{x}(t)}{\partial \hat{p}_i \partial \hat{p}_j}, \quad i, j = 1, 2, \dots, 2n+1 \quad (9.48)$$

$$\frac{d}{dt} \left(\frac{\partial \hat{x}(t)}{\partial \hat{a}_k} \right) = \hat{A} \frac{\partial \hat{x}(t)}{\partial \hat{a}_k} - G_k \hat{y}(t), \quad k = 1, 2, \dots, n \quad (9.49)$$

$$\frac{d}{dt} \left(\frac{\partial \hat{x}(t)}{\partial \hat{b}_k} \right) = \hat{A} \frac{\partial \hat{x}(t)}{\partial \hat{b}_k} + G_k u(t - \hat{\theta}), \quad k = 1, 2, \dots, n \quad (9.50)$$

$$\frac{d}{dt} \left(\frac{\partial \hat{x}(t)}{\partial \hat{\theta}} \right) = \hat{A} \frac{\partial \hat{x}(t)}{\partial \hat{\theta}} + B \frac{\partial u(t - \hat{\theta})}{\partial \hat{\theta}} \quad (9.51)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{a}_l \partial \hat{a}_k} \right) = \hat{A} \frac{\partial^2 \hat{x}(t)}{\partial \hat{a}_l \partial \hat{a}_k} - G_l \frac{\partial \hat{y}(t)}{\partial \hat{a}_k} - G_k \frac{\partial \hat{y}(t)}{\partial \hat{a}_l}, \quad k = 1, 2, \dots, n \quad l = k, k+1, \dots, n \quad (9.52)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{b}_l \partial \hat{a}_k} \right) = \hat{A} \frac{\partial^2 \hat{x}(t)}{\partial \hat{b}_l \partial \hat{a}_k} - G_k \frac{\partial \hat{y}(t)}{\partial \hat{b}_l}, \quad k = 1, 2, \dots, n \quad l = 1, 2, \dots, n \quad (9.53)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta} \partial \hat{a}_k} \right) = \hat{A} \frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta} \partial \hat{a}_k} - G_k \frac{\partial \hat{y}(t)}{\partial \hat{\theta}}, \quad k = 1, 2, \dots, n \quad (9.54)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{b}_l \partial \hat{b}_k} \right) = 0, \quad k = 1, 2, \dots, n \quad l = k, k+1, \dots, n \quad (9.55)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta} \partial \hat{b}_k} \right) = \hat{A} \frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta} \partial \hat{b}_k} + G_k \frac{\partial u(t - \hat{\theta})}{\partial \hat{\theta}}, \quad k = 1, 2, \dots, n \quad (9.56)$$

$$\frac{d}{dt} \left(\frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta}^2} \right) = \hat{A} \frac{\partial^2 \hat{x}(t)}{\partial \hat{\theta}^2} + B \frac{\partial^2 u(t - \hat{\theta})}{\partial \hat{\theta}^2} \quad (9.57)$$

$$\frac{\partial^2 \hat{x}(t)}{\partial \phi \partial \phi} = \frac{\partial^2 \hat{x}(t)}{\partial \phi \partial \phi}, \quad \phi, \varphi = \hat{a}_k, \hat{b}_k, \hat{\theta} \quad (9.58)$$

where

$$G_k = \begin{bmatrix} 0 & 0 & \cdots & \underbrace{1 \ 0 \ \cdots \ 0}_k \end{bmatrix}^T$$

has 1 for the $n - k + 1$ th component and 0 for the others. The initial values of all the derivatives of the state $\hat{x}(t)$ are zero because the parameter of \hat{p} does not affect the initial values of the state. The derivative of $u(t - \hat{\theta})$ in (9.51), (9.56) and (9.57) with respect to $\hat{\theta}$ is calculated using

$$\frac{\partial u(t - \hat{\theta})}{\partial \hat{\theta}} = - \frac{du(t - \hat{\theta})}{dt}, \quad \frac{\partial^2 u(t - \hat{\theta})}{\partial \hat{\theta}^2} = \frac{\partial^2 u(t - \hat{\theta})}{\partial t^2} \quad (9.59)$$

With a numerical derivative, (9.59) can be calculated from the given process input.

In summary, $\hat{x}(t_i)$ and all the derivatives of the state are calculated for the given $\hat{p}(j-1)$ by selecting them at every t_i while continuously solving the ordinary differential equations (9.39) and (9.49)–(9.57) simultaneously. Then, it is straightforward to calculate the updated parameters $\hat{p}(j)$ from (9.42). The whole procedure is repeated until the parameters converge.

Example 9.5

The following continuous-time process with time delay is simulated to confirm the identification performance of the prediction error identification method for the continuous-time differential equation model:

$$\frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (9.60)$$

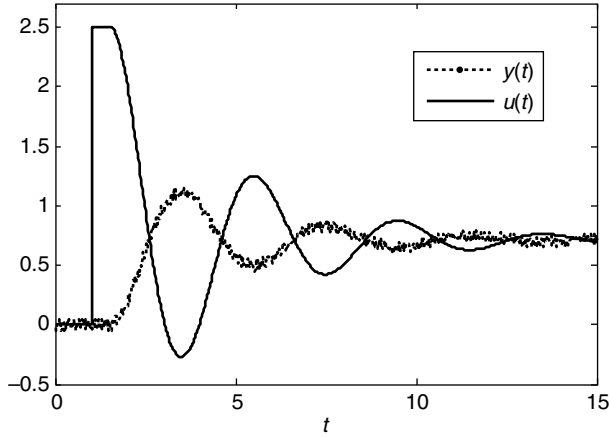


Figure 9.13 Process input and output activated by a P controller.

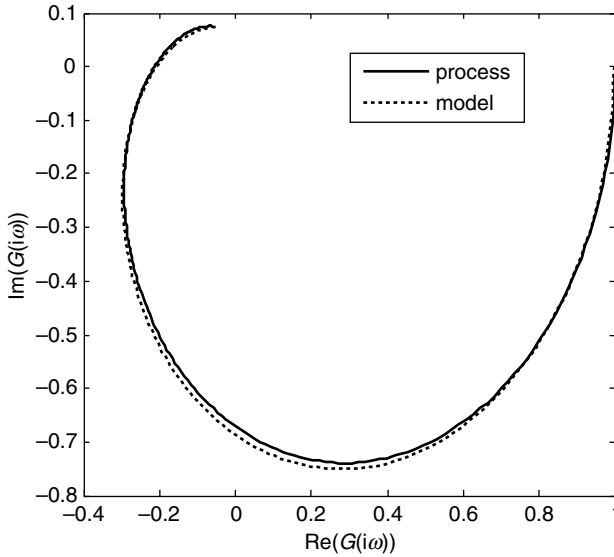


Figure 9.14 Identification results of the prediction error identification method.

The process is activated by a P controller for which the setpoint is changed at $t = 1$. The process output is contaminated by uniformly distributed random noise between -0.05 and 0.05 . The sampling time is 0.02 .

Figures 9.13 and 9.14 show the activated process input and output and the identification result. Figure 9.14 confirms that the prediction error method provides acceptable performance under the circumstance of the measurement noise.

The MATLAB code to solve the estimation problem in Example 9.5 and draw Figure 9.14 are shown in Tables 9.9 and 9.10 respectively. Table 9.9 uses numerical derivatives to calculate the partial derivatives $\partial^2 V(\hat{p}(j-1))/\partial \hat{p}^2$ and $\partial V(\hat{p}(j-1))/\partial \hat{p}$ in (9.42).

Table 9.9 MATLAB code to solve the estimation problem using the prediction error method in Example 9.5.

```

                                continuous_PEM1.m

clear; delt=0.02; tf=15; ys=0.0; n=round(tf/delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0 ; 0]; y=0.0; u_data=zeros(1,1001);
iter=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.1;
for i=1:n
    t=i*delt; yy(i)=y+noise(i); tt(i)=t; yys(i)=ys;
    if (t>1) ys=1.0; end
    u=2.5*(ys-yy(i));
    for j=1:1000
        u_data(j)=u_data(j+1);
    end
    uu(i)=u; u_data(1001)=u;
    [x,y]=g_continuous_PEM1(x,delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(tt,yy,tt,uu); legend('y(t)','u(t)');
% starting PEM
a1=3.5; a2=5.0; b1=0.0; b2=1.5; delay=0.0; %initial values of model
delta=0.0001; del_delay=delt; %interval for the numerical derivatives
alpha=10.0; index_update=1;
Pb=[a1 a2 b1 b2 delay]';
E=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay);
fprintf('iteration=%2d a1=%6.3f a2=%6.3f b1=%6.3f b2=%6.3f delay=%6.3f\n',iter,Pb(1),Pb(2),Pb(3),Pb(4),Pb(5),E);

while(1)
    if(index_update==1)
        a1=Pb(1); a2=Pb(2); b1=Pb(3); b2=Pb(4); delay=Pb(5);
        E=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay); %object function
        E_a1=object_PEM1(uu,yy,delt,tf,a1+delta,a2,b1,b2,delay);
        E_a2=object_PEM1(uu,yy,delt,tf,a1,a2+delta,b1,b2,delay);
        E_b1=object_PEM1(uu,yy,delt,tf,a1,a2,b1+delta,b2,delay);
        E_b2=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2+delta,delay);
        E_delay=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay+del_delay);

        E_a1_a1=object_PEM1(uu,yy,delt,tf,a1+2*delta,a2,b1,b2,delay);
        E_a2_a2=object_PEM1(uu,yy,delt,tf,a1,a2+2*delta,b1,b2,delay);
        E_b1_b1=object_PEM1(uu,yy,delt,tf,a1,a2,b1+2*delta,b2,delay);
        E_b2_b2=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2+2*delta,delay);
        E_delay_delay=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay+2*del_delay);

        E_a1_a2=object_PEM1(uu,yy,delt,tf,a1+delta,a2+delta,b1,b2,delay);
        E_a1_b1=object_PEM1(uu,yy,delt,tf,a1+delta,a2,b1+delta,b2,delay);
        E_a1_b2=object_PEM1(uu,yy,delt,tf,a1+delta,a2,b1,b2+delta,delay);

```

(continued)

Table 9.9 (Continued)

```

E_a1_delay=object_PEM1(uu,yy,delt,tf,a1+delta,a2,b1,b2,delay
+del_delay);

    E_a2_b1=object_PEM1(uu,yy,delt,tf,a1,a2+delta,b1+delta,b2,delay);
    E_a2_b2=object_PEM1(uu,yy,delt,tf,a1,a2+delta,b1,b2+delta,delay);

E_a2_delay=object_PEM1(uu,yy,delt,tf,a1,a2+delta,b1,b2,delay
+del_delay);

    E_b1_b2=object_PEM1(uu,yy,delt,tf,a1,a2,b1+delta,b2+delta,delay);

E_b1_delay=object_PEM1(uu,yy,delt,tf,a1,a2,b1+delta,b2,delay
+del_delay);
    E_b2_delay=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2+delta,delay
+del_delay);

dV(1,1)=(E_a1-E)/delta;%dV/dp1
dV(2,1)=(E_a2-E)/delta;%dV/dp2
dV(3,1)=(E_b1-E)/delta;%dV/dp3
dV(4,1)=(E_b2-E)/delta;%dV/dp4
dV(5,1)=(E_delay-E)/del_delay;%dV/dp5

ddV(1,1)=(E_a1_a1-2*E_a1+E)/delta^2;%d^2V/dp1^2
ddV(2,2)=(E_a2_a2-2*E_a2+E)/delta^2;
ddV(3,3)=(E_b1_b1-2*E_b1+E)/delta^2;
ddV(4,4)=(E_b2_b2-2*E_b2+E)/delta^2;
ddV(5,5)=(E_delay_delay-2*E_delay+E)/del_delay^2;

ddV(1,2)=(E_a1_a2-E_a1-E_a2+E)/delta^2;%d^2V/dp1dp2
ddV(1,3)=(E_a1_b1-E_a1-E_b1+E)/delta^2;
ddV(1,4)=(E_a1_b2-E_a1-E_b2+E)/delta^2;
ddV(1,5)=(E_a1_delay-E_a1-E_delay+E)/(delta*del_delay);

ddV(2,3)=(E_a2_b1-E_a2-E_b1+E)/delta^2;
ddV(2,4)=(E_a2_b2-E_a2-E_b2+E)/delta^2;
ddV(2,5)=(E_a2_delay-E_a2-E_delay+E)/(delta*del_delay);

ddV(3,4)=(E_b1_b2-E_b1-E_b2+E)/delta^2;
ddV(3,5)=(E_b1_delay-E_b1-E_delay+E)/(delta*del_delay);

ddV(4,5)=(E_b2_delay-E_b2-E_delay+E)/(delta*del_delay);
ddV(2,1)=ddV(1,2); ddV(3,1)=ddV(1,3); ddV(4,1)=ddV(1,4); ddV(5,1)
=ddV(1,5);
ddV(3,2)=ddV(2,3); ddV(4,2)=ddV(2,4); ddV(5,2)=ddV(2,5);
ddV(4,3)=ddV(3,4); ddV(5,3)=ddV(3,5); ddV(5,4)=ddV(4,5);
end
P=Pb-inv(ddV+alpha*eye(5))*dV;

```

Table 9.9 (Continued)

```

if (P(5)<0.0) P(5)=0.0; end
if (P(5)>2.0) P(5)=2.0; end
a1=P(1); a2=P(2); b1=P(3); b2=P(4); delay=P(5);
[E_new yy_m]=object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay);
if (E_new<E & P(5)>=0.0)
    alpha=alpha/2.0; Pb=P;
    iter=iter+1; index_update=1;
    fprintf('iteration=%2d a1=%6.3f a2=%6.3f b1=%6.3f b2=%6.3f delay=%6.3f E=%6.3f\n',iter,P(1),P(2),P(3),P(4),P(5),E);
else
    index_update=0;
    alpha=alpha*1.5;
end
if (iter==20 | alpha>10.0^10) break; end
end
figure(2); plot(tt,yy,'c',tt,yy_m,'k'); legend('process','model');

```

g_continuous_PEM1.m

```

function [next_x,y]=g_continuous_PEM1(x,delt,u,a1,a2,b1,b2,delay);
subdelt=0.005; n=round(delt/subdelt);
A=[0 -a2; 1 -a1]; B=[b2; b1]; C=[0 1];
delay_k=round(delay/delt);
for i=1:n
    dx=A*x+B*u(1000-delay_k);
    x=x+dx*subdelt;
end
next_x=x; y=C*x;
return

```

object_PEM1.m

```

function [V yy_m] = object_PEM1(uu,yy,delt,tf,a1,a2,b1,b2,delay)
x=[0; 0]; y=0.0;
u_data=zeros(1,1001); n=round(tf/delt);
for i=1:n
    t=i*delt; yy_m(i)=y; tt(i)=t;
    for j=1:1000
        u_data(j)=u_data(j+1);
    end
    u_data(1001)=uu(i);
    [x,y]=m_continuous_PEM1(x,delt,u_data,a1,a2,b1,b2,delay);
end
s=0;
for i=1:length(yy)
    s=s+(yy(i)-yy_m(i))^2*delt;
end
V=s;

```

(continued)

Table 9.9 (Continued)

```

                                m_continuous_PEM1.m

function [next_x,y]=m_continuous_PEM1(x,delt,u,a1,a2,b1,b2,delay);
    subdelt=0.005; n=round(delt/subdelt);
    A=[0 -a2; 1 -a1]; B=[b2; b1]; C=[0 1];
    delay_k=round(delay/delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*subdelt;
    end
    next_x=x; y=C*x;
return

```

```

                                >> continuous_PEM1

iteration= 0 a1= 3.500 a2= 5.000 b1= 0.000 b2= 1.500 delay= 0.000 E= 3.936
iteration= 1 a1= 3.520 a2= 4.932 b1=-0.068 b2= 1.646 delay= 0.119 E= 3.936
iteration= 2 a1= 3.554 a2= 4.777 b1=-0.178 b2= 1.930 delay= 0.367 E= 3.492
iteration= 3 a1= 3.609 a2= 4.440 b1=-0.272 b2= 2.415 delay= 0.693 E= 2.625
iteration= 4 a1= 3.766 a2= 3.854 b1=-0.247 b2= 2.921 delay= 0.825 E= 1.420
iteration= 5 a1= 4.085 a2= 3.271 b1=-0.190 b2= 3.047 delay= 0.816 E= 0.532

```

Table 9.11 shows a typical convergence pattern of the prediction error identification method. α is updated according to Marquardt's compromise with the initial value of $\alpha = 1.0$. For details, refer to Chapter 2. Considering the confirmed robustness of the Levenberg–Marquardt method and extensive simulation results for various initial settings, it is concluded that the nonlinear

Table 9.10 MATLAB code to draw Figure 9.10.

```

                                nyquist_continuous_PEM1.m

del_w=0.01; wmax=pi; n=round(wmax/del_w);
for i=1:n
    w=(i-1)*del_w; j=complex(0,1); s=j*w;
    gjw=exp(-0.5*s)/(s+1)^2;
    gjw_model=(b1*s+b2)*exp(-delay*s)/(s^2+a1*s+a2);
    Re(i)=real(gjw); Im(i)=imag(gjw);
    Re_model(i)=real(gjw_model);
    Im_model(i)=imag(gjw_model);
end
figure(1); plot(Re,Im,Re_model,Im_model,':');

```

```

                                command window

>> nyquist_continuous_PEM1

```

Table 9.11 Typical convergence pattern for the prediction error identification method.

iteration=	0	a1= 3.500	a2= 5.000	b1= 0.000	b2= 1.500	delay= 0.000	E= 3.936
iteration=	1	a1= 3.520	a2= 4.932	b1=-0.068	b2= 1.646	delay= 0.119	E= 3.936
iteration=	2	a1= 3.554	a2= 4.777	b1=-0.178	b2= 1.930	delay= 0.367	E= 3.492
iteration=	3	a1= 3.609	a2= 4.440	b1=-0.272	b2= 2.415	delay= 0.693	E= 2.625
iteration=	4	a1= 3.766	a2= 3.854	b1=-0.247	b2= 2.921	delay= 0.825	E= 1.420
iteration=	5	a1= 4.085	a2= 3.271	b1=-0.190	b2= 3.047	delay= 0.816	E= 0.532
iteration=	6	a1= 4.398	a2= 2.924	b1=-0.145	b2= 2.845	delay= 0.775	E= 0.201
iteration=	7	a1= 4.629	a2= 2.616	b1=-0.119	b2= 2.606	delay= 0.734	E= 0.090
iteration=	8	a1= 4.733	a2= 2.415	b1=-0.124	b2= 2.436	delay= 0.696	E= 0.037
iteration=	9	a1= 4.714	a2= 2.315	b1=-0.157	b2= 2.346	delay= 0.666	E= 0.025
iteration=	10	a1= 4.568	a2= 2.220	b1=-0.206	b2= 2.252	delay= 0.643	E= 0.025
iteration=	11	a1= 4.233	a2= 2.058	b1=-0.175	b2= 2.086	delay= 0.632	E= 0.024
iteration=	12	a1= 3.380	a2= 1.648	b1=-0.114	b2= 1.666	delay= 0.592	E= 0.023
iteration=	13	a1= 2.181	a2= 1.067	b1=-0.149	b2= 1.070	delay= 0.446	E= 0.019
iteration=	14	a1= 1.860	a2= 0.934	b1=-0.129	b2= 0.931	delay= 0.346	E= 0.015
iteration=	15	a1= 1.861	a2= 0.933	b1=-0.130	b2= 0.930	delay= 0.343	E= 0.012
iteration=	16	a1= 1.861	a2= 0.933	b1=-0.130	b2= 0.930	delay= 0.340	E= 0.012

optimization method gives acceptable robustness if the initial settings are roughly close to the true values.

Example 9.6

Repeat Example 9.5 with a sampling time of 0.1.

Figures 9.15 and 9.16 show the activated process input and output and the identification result. Figure 9.16 confirms that the prediction error method provides acceptable performance under the circumstance of the measurement noises.

The MATLAB code to solve the estimation problem in Example 9.6 and draw Figure 9.16 are shown in Tables 9.12 and 9.13 respectively.

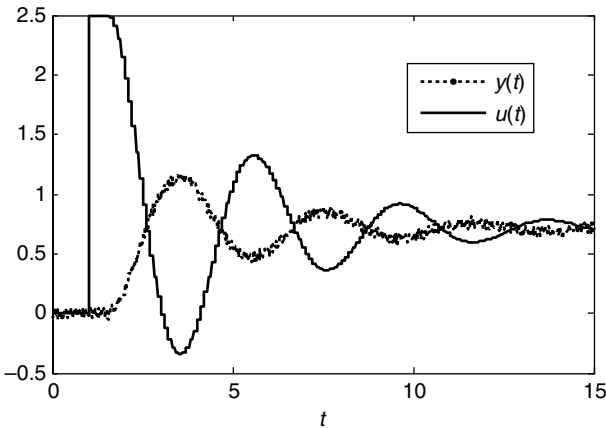


Figure 9.15 Process input and output activated by a P controller with a sampling time of 0.1.

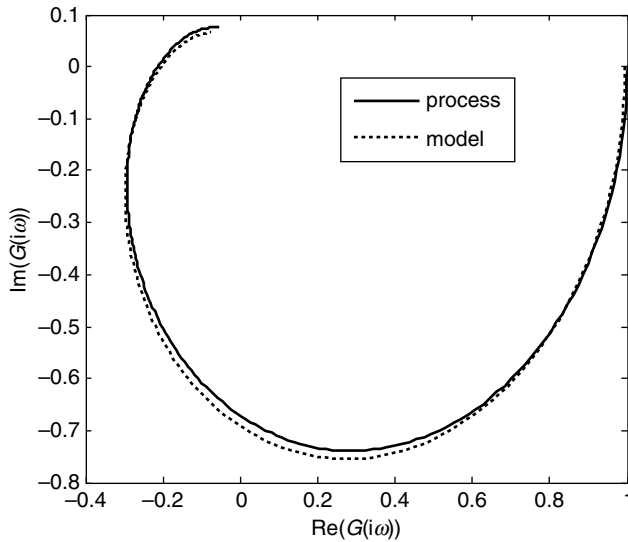


Figure 9.16 Identification results of the prediction error identification method.

Table 9.12 MATLAB code to solve the estimation problem using the prediction error method in Example 9.6.

```

continuous_PEM2.m

clear;
delt=0.1; sub_delt=0.02; tf=15; tref=-delt+sub_delt; ys=0.0;
n=round(tf/sub_delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0 ; 0]; y=0.0; u_data=zeros(1,1001); u=0.0;
iter=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.1; k=1;
for i=1:n
    t=i*sub_delt; yyy(i)=y+noise(i); ttt(i)=t; yys(i)=ys;
    if (t>1) ys=1.0; end
    if (abs(t-(tref+delt))<0.001) tref=t; u=2.5*(ys-yyy(i)); uu(k)=u;
yy(k)=yyy(i); tt(k)=t; k=k+1; end
    for j=1:1000
        u_data(j)=u_data(j+1);
    end
    uuu(i)=u; u_data(1001)=u;
    [x,y]=g_continuous_PEM2(x,sub_delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(ttt,yyy,ttt,uuu); legend('y(t)','u(t)');
% starting PEM
a1=3.5; a2=5.0; b1=0.0; b2=1.5; delay=0.0; %initial values of model
delta=0.0001; del_delay=sub_delt; %interval for the numerical
derivatives

```


Table 9.12 (Continued)

```

alpha=10.0; index_update=1;
Pb=[a1 a2 b1 b2 delay]';
E=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay);
fprintf(' iteration=%2d a1=%6.3f a2=%6.3f b1=%6.3f b2=%6.3f delay=%6.3f
E=%6.3f\n', iter, Pb(1), Pb(2), Pb(3), Pb(4), Pb(5), E);

while(1)
    if(index_update==1)

        a1=Pb(1); a2=Pb(2); b1=Pb(3); b2=Pb(4); delay=Pb(5);

        E=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay); %object
function
        E_a1=object_PEM2(uu,yy,sub_delt,tt,a1+delta,a2,b1,b2,delay);
        E_a2=object_PEM2(uu,yy,sub_delt,tt,a1,a2+delta,b1,b2,delay);
        E_b1=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1+delta,b2,delay);
        E_b2=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2+delta,delay);
        E_delay=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay+
del_delay);

        E_a1_a1=object_PEM2(uu,yy,sub_delt,tt,a1+2*delta,a2,b1,b2,delay);
        E_a2_a2=object_PEM2(uu,yy,sub_delt,tt,a1,a2+2*delta,b1,b2,delay);
        E_b1_b1=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1+2*delta,b2,delay);
        E_b2_b2=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2+2*delta,delay);
        E_delay_delay=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay+
2*del_delay);

        E_a1_a2=object_PEM2(uu,yy,sub_delt,tt,a1+delta,a2+delta,b1,b2,
delay);
        E_a1_b1=object_PEM2(uu,yy,sub_delt,tt,a1+delta,a2,b1+delta,b2,
delay);
        E_a1_b2=object_PEM2(uu,yy,sub_delt,tt,a1+delta,a2,b1,b2+delta,
delay);
        E_a1_delay=object_PEM2(uu,yy,sub_delt,tt,a1+delta,a2,b1,b2,delay+
del_delay);

        E_a2_b1=object_PEM2(uu,yy,sub_delt,tt,a1,a2+delta,b1+delta,b2,
delay);
        E_a2_b2=object_PEM2(uu,yy,sub_delt,tt,a1,a2+delta,b1,b2+delta,
delay);
        E_a2_delay=object_PEM2(uu,yy,sub_delt,tt,a1,a2+delta,b1,b2,delay+
del_delay);

        E_b1_b2=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1+delta,b2+delta,
delay);
        E_b1_delay=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1+delta,b2,delay+
del_delay);

```

(continued)

Table 9.12 (Continued)

```

    E_b2_delay=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2+delta,delay+
del_delay);

    dV(1,1)=(E_a1-E)/delta;%dV/dp1
    dV(2,1)=(E_a2-E)/delta;%dV/dp2
    dV(3,1)=(E_b1-E)/delta;%dV/dp3
    dV(4,1)=(E_b2-E)/delta;%dV/dp4
    dV(5,1)=(E_delay-E)/del_delay;%dV/dp5

    ddV(1,1)=(E_a1_a1-2*E_a1+E)/delta^2;%d^2V/dp1^2
    ddV(2,2)=(E_a2_a2-2*E_a2+E)/delta^2;
    ddV(3,3)=(E_b1_b1-2*E_b1+E)/delta^2;
    ddV(4,4)=(E_b2_b2-2*E_b2+E)/delta^2;
    ddV(5,5)=(E_delay_delay-2*E_delay+E)/del_delay^2;

    ddV(1,2)=(E_a1_a2-E_a1-E_a2+E)/delta^2;%d^2V/dp1dp2
    ddV(1,3)=(E_a1_b1-E_a1-E_b1+E)/delta^2;
    ddV(1,4)=(E_a1_b2-E_a1-E_b2+E)/delta^2;
    ddV(1,5)=(E_a1_delay-E_a1-E_delay+E)/(delta*del_delay);

    ddV(2,3)=(E_a2_b1-E_a2-E_b1+E)/delta^2;
    ddV(2,4)=(E_a2_b2-E_a2-E_b2+E)/delta^2;
    ddV(2,5)=(E_a2_delay-E_a2-E_delay+E)/(delta*del_delay);

    ddV(3,4)=(E_b1_b2-E_b1-E_b2+E)/delta^2;
    ddV(3,5)=(E_b1_delay-E_b1-E_delay+E)/(delta*del_delay);

    ddV(4,5)=(E_b2_delay-E_b2-E_delay+E)/(delta*del_delay);

    ddV(2,1)=ddV(1,2); ddV(3,1)=ddV(1,3); ddV(4,1)=ddV(1,4); ddV(5,1)=
ddV(1,5);
    ddV(3,2)=ddV(2,3); ddV(4,2)=ddV(2,4); ddV(5,2)=ddV(2,5);
    ddV(4,3)=ddV(3,4); ddV(5,3)=ddV(3,5); ddV(5,4)=ddV(4,5);
end
P=Pb-inv(ddV+alpha*eye(5))*dV;
if (P(5)<0.0) P(5)=0.0; end
if (P(5)>2.0) P(5)=2.0; end
a1=P(1); a2=P(2); b1=P(3); b2=P(4); delay=P(5);
[E_new yy_m]=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay);
if (E_new<E & P(5)>=0.0)
    alpha=alpha/2.0; Pb=P;
    iter=iter+1; index_update=1;
    fprintf('iteration=%2d a1=%6.3f a2=%6.3f b1=%6.3f b2=%6.3f
delay=%6.3f E=%6.3f\n',iter,P(1),P(2),P(3),P(4),P(5),E);
else
    index_update=0;
    alpha=alpha*1.5;
end
if (iter==20 | alpha>10.0^10) break; end

```

Table 9.12 (Continued)

```
end
figure(2); plot(tt,yy,'c',tt,yy_m,'k'); legend('process','model');
```

g_continuous_PEM2.m

```
function [next_x,y]=g_continuous_PEM2(x,sub_delt,u,a1,a2,b1,b2,
delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return
```

object_PEM2.m

```
function [V yy_m]=object_PEM2(uu,yy,sub_delt,tt,a1,a2,b1,b2,delay)
    x=[0; 0]; y=0.0; u_data=zeros(1,1001);
    k=1; m=length(tt); tf=tt(m); n=round(tf/sub_delt)+1; s=0.0;
    for i=1:n
        t=i*sub_delt;
        for j=1:1000
            u_data(j)=u_data(j+1);
        end
        if(k<=m & abs(t-tt(k))<sub_delt/2.1)
            if(k<m) s=s+(yy(k)-y)^2*(tt(k+1)-tt(k)); end
            yy_m(k)=y; u=uu(k); k=k+1;
        end
        u_data(1001)=u;
        [x,y]=m_continuous_PEM2(x,sub_delt,u_data,a1,a2,b1,b2,delay);
    end
    V=s;
```

m_continuous_PEM2.m

```
function [next_x,y]=m_continuous_PEM2(x,sub_delt,u,a1,a2,b1,b2,
delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return
```

(continued)

Table 9.12 (Continued)

command window	
<hr/>	
>> continuous_PEM2	
iteration= 0	a1= 3.500 a2= 5.000 b1= 0.000 b2= 1.500 delay= 0.000 E= 3.912
iteration= 1	a1= 3.521 a2= 4.932 b1=-0.067 b2= 1.644 delay= 0.119 E= 3.912
iteration= 2	a1= 3.556 a2= 4.779 b1=-0.177 b2= 1.926 delay= 0.368 E= 3.473
iteration= 3	a1= 3.617 a2= 4.443 b1=-0.280 b2= 2.400 delay= 0.648 E= 2.616
iteration= 4	a1= 3.775 a2= 3.855 b1=-0.251 b2= 2.912 delay= 0.847 E= 1.467
iteration= 5	a1= 4.104 a2= 3.263 b1=-0.193 b2= 3.034 delay= 0.804 E= 0.546
iteration= 6	a1= 4.422 a2= 2.901 b1=-0.167 b2= 2.824 delay= 0.751 E= 0.209
iteration= 7	a1= 4.646 a2= 2.602 b1=-0.098 b2= 2.591 delay= 0.730 E= 0.093
iteration= 8	a1= 4.745 a2= 2.398 b1=-0.110 b2= 2.420 delay= 0.679 E= 0.041
iteration= 9	a1= 4.714 a2= 2.296 b1=-0.164 b2= 2.327 delay= 0.648 E= 0.029
iteration=10	a1= 4.557 a2= 2.199 b1=-0.197 b2= 2.230 delay= 0.630 E= 0.028
iteration=11	a1= 4.164 a2= 2.002 b1=-0.215 b2= 2.029 delay= 0.597 E= 0.028
iteration=12	a1= 3.096 a2= 1.496 b1=-0.143 b2= 1.509 delay= 0.534 E= 0.026
iteration=13	a1= 2.218 a2= 1.083 b1=-0.132 b2= 1.085 delay= 0.423 E= 0.020
iteration=14	a1= 1.678 a2= 0.840 b1=-0.156 b2= 0.834 delay= 0.266 E= 0.015
iteration=15	a1= 1.762 a2= 0.897 b1=-0.041 b2= 0.890 delay= 0.413 E= 0.013
iteration=16	a1= 1.774 a2= 0.893 b1=-0.145 b2= 0.888 delay= 0.272 E= 0.013

The simulations of Examples 9.5 and 9.6 exemplify that the prediction error identification method shows good results for small and large sampling times. Also, it can incorporate an input time delay efficiently. Previous continuous-time approaches using integral or delta transforms cannot be applied to large sampling times because the numerical integration or the derivative of the process output is not accurate for a large sampling time.

Table 9.13 MATLAB code to draw Figure 9.12.

nyquist_continuous_PEM2.m	
<hr/>	
del_w=0.01; wmax=pi; n=round(wmax/del_w);	
for i=1:n	
w=(i-1)*del_w; j=complex(0,1); s=j*w;	
gjwt=exp(-0.5*s)/(s+1)^2;	
gjwt_model=(b1*s+b2)*exp(-delay*s)/(s^2+a1*s+a2);	
Re(i)=real(gjwt); Im(i)=imag(gjwt);	
Re_model(i)=real(gjwt_model);	
Im_model(i)=imag(gjwt_model);	
end	
figure(1); plot(Re, Im, Re_model, Im_model, ' : ');	
<hr/>	
command window	
<hr/>	
>> nyquist_continuous_PEM2	

9.2.3 Concluding Remarks

A continuous-time prediction error identification method was introduced for continuous-time processes with time delay. This has advantages compared with previous continuous-time and discrete-time identification methods, since it can incorporate small, large and irregular sampling times and input time delays as well. However, solving the nonlinear optimization problem using the Levenberg–Marquardt method is a disadvantage compared with other process identification methods using the least-squares method.

Problems

- 9.1 Activate the process using the following PI controller and estimate the model using the identification method in Section 9.1 for the case of an initially steady state. Compare the Nyquist plot of the model with that of the process.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = -0.1 \frac{du(t-0.1)}{dt} + u(t-0.1)$$

$$u(t) = 1.5(y_s(t) - y(t)) + \frac{1.5}{3.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t < 0$$

$$y_s = 1.0 \quad \text{for } t \geq 0 \quad \text{and} \quad y_s = 0.0 \quad \text{for } t < 0$$

- 9.2 Tune the PID controller using the ITAE-2 tuning rule for the model obtained in Problem 9.1 and simulate the control performance.
- 9.3 Determine if you can apply the identification method in Section 9.1 for the case of an initially unsteady state. Justify your conclusion.
- 9.4 Activate the process using the following PID controller and estimate the model using the identification method in Section 9.1 for the case of an initially unsteady state. Compare the Nyquist plot of the model with that of the process.

$$\frac{d^3 y(t)}{dt^3} + 3 \frac{d^2 y(t)}{dt^2} + 3 \frac{dy(t)}{dt} + y(t) = -0.1 \frac{du(t-0.1)}{dt} + u(t-0.1)$$

$$u(t) = 1.5(y_s(t) - y(t)) + \frac{1.5}{3.0} \int_0^t (y_s(\tau) - y(\tau)) d\tau + 0.5 \frac{d(y_s(t) - y(t))}{dt} - 0.3$$

$$\left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = 0, \quad \left. \frac{dy(t)}{dt} \right|_{t=0} = -0.5, \quad y(0) = -0.3, \quad u(t) = 0, \quad t < 0$$

$$y_s = 1.0 \quad \text{for } t \geq 5 \quad \text{and} \quad y_s = 0.0 \quad \text{for } t < 0$$

- 9.5 Tune the PID controller using the ITAE-2 tuning rule for the model obtained in Problem 9.4 and simulate the control performance.

- 9.6 Activate the virtual process of Process 3 (refer to the Appendix for details) using a biased-relay and obtain the model using the two identification methods in Section 9.1.
- 9.7 Tune the PID controller using the ITAE-2 tuning rule for the model obtained in Problem 9.6 and show the control performance for the virtual process.
- 9.8 Estimate the model using the prediction error identification method in Section 9.2 for the activated process output and input data of Problem 9.1. Compare the Nyquist plot of the model and that of the process.
- 9.9 Activate the process of Problem 9.1 with a sampling time of 0.2 and estimate the model using the prediction error identification method in Section 9.2 for the activated process output and input data. Compare the Nyquist plot of the model and that of the process.
- 9.10 Activate the virtual process of Process 3 (refer to the Appendix for details) using a biased-relay and obtain the model using the prediction error identification method in Section 9.2. In the case, determine the initial estimates using the Fourier transform or the modified Fourier transform.
- 9.11 Tune the PID controller using the ITAE-2 tuning rule for the model obtained in Problem 9.10 and show the control performance for the virtual process.

References

- Sung, S.W., Lee, I. and Lee, J. (1998) New process identification method for automatic design of PID controllers. *Automatica*, **34**, 513.
- Sung, S.W. and Lee, I. (1999) On-line process identification and PID controller autotuning. *Korean Journal of Chemical Engineering*, **16**, 45.
- Sung, S.W. and Lee, I. (2001) Prediction error identification method for continuous-time processes with time delay. *Industrial & Engineering Chemistry Research*, **40**, 5743.
- Sung, S.W., Lee, S.Y., Kwak, H.J. and Lee, I. (2001) Continuous-time subspace system identification method. *Industrial & Engineering Chemistry Research*, **40**, 2886–2896.

10

Process Identification Methods for Discrete-Time Difference Equation Models

10.1 Prediction Models: Autoregressive Exogenous Input Model and Output Error Model

Prediction models are used to predict the future process output on the basis of the past process input and/or past process output. In this section, two types of discrete-time prediction model are introduced. One is the ARX model and the other is the output error (OE) model. Various discrete-time difference models have been used for modeling of a process. The most useful in process systems engineering are the ARX model and the OE model. Other discrete-time difference models, such as the autoregressive, moving-average, exogenous input (ARMAX) model and the autoregressive, integrated-moving-average, exogenous input (ARIMAX) model, are useful for noisy environments. All the models for noisy processes assume that the noise is white noise. But noise in process systems engineering is small and uncertainties such as disturbances and noises are also quite different from white noise. Also, the models for noisy environments have complicated structures compared with an ARX or OE model. So, the ARX model and OE model are more useful for most cases in process systems engineering than the other more complicated models.

10.1.1 Autoregressive Exogenous Input Model

The ARX model has the following model structure:

$$\begin{aligned} \hat{y}(k\Delta t) = & -\hat{a}_1 y((k-1)\Delta t) - \hat{a}_2 y((k-2)\Delta t) - \cdots - \hat{a}_n y((k-n)\Delta t) \\ & + \hat{b}_1 u((k-1-\hat{d})\Delta t) + \hat{b}_2 u((k-2-\hat{d})\Delta t) + \cdots + \hat{b}_n u((k-n-\hat{d})\Delta t) + \hat{B} \end{aligned} \quad (10.1)$$

where Δt is the sampling time. \hat{d} is the number of the sampling time corresponding to the time delay. That is, $\hat{d}\Delta t$ is the time delay. n is the model order. $y(k\Delta t)$ and $u(k\Delta t)$ are the measurements of the process output and the process input at the k th sampling. The model output $\hat{y}(k\Delta t)$ is the predicted process output at the k th sampling. The coefficients of \hat{d} , \hat{a}_i ($i = 1, 2, \dots, n$) and

\hat{b}_i ($i = 1, 2, \dots, n$) are the model parameters of the ARX model. As shown in (10.1), $\hat{y}(k\Delta t)$ depends on the past process output $y((k-1)\Delta t), y((k-2)\Delta t), \dots, y((k-n)\Delta t)$ and the past process input $u((k-1-\hat{d})\Delta t), u((k-2-\hat{d})\Delta t), \dots, u((k-n-\hat{d})\Delta t)$. It should be noted that a one-step-before process output $y((k-1)\Delta t)$ is needed to estimate the model output $\hat{y}(k\Delta t)$. That is, the ARX model can only predict a one-step-ahead process output. So, it can be said that the ARX model is a one-step-ahead predictor.

10.1.2 Output Error Model

The OE model has the following structure:

$$\begin{aligned} \hat{y}(k\Delta t) = & -\hat{a}_1\hat{y}((k-1)\Delta t) - \hat{a}_2\hat{y}((k-2)\Delta t) - \dots - \hat{a}_n\hat{y}((k-n)\Delta t) \\ & + \hat{b}_1u((k-1-\hat{d})\Delta t) + \hat{b}_2u((k-2-\hat{d})\Delta t) + \dots + \hat{b}_nu((k-n-\hat{d})\Delta t) + \hat{B} \end{aligned} \quad (10.2)$$

where Δt is the sampling time. \hat{d} is the number of the sampling time corresponding to the time delay. That is, $\hat{d}\Delta t$ is the time delay. n is the model order. $u(k\Delta t)$ is the measurement of the process input at the k th sampling. The model output of $\hat{y}(k\Delta t)$ is the predicted process output at the k th sampling. The coefficients of \hat{d}, \hat{a}_i ($i = 1, 2, \dots, n$) and \hat{b}_i ($i = 1, 2, \dots, n$) are the model parameters of the OE model. As shown in (10.2), $\hat{y}(k\Delta t)$ depends on the past model output $\hat{y}((k-1)\Delta t), \hat{y}((k-2)\Delta t), \dots, \hat{y}((k-n)\Delta t)$ and the past process input $u((k-1-\hat{d})\Delta t), u((k-2-\hat{d})\Delta t), \dots, u((k-n-\hat{d})\Delta t)$. It should be noted that all the model output in the future can be estimated only if the process input is known. So, it can be said that the OE model is a multistep-ahead predictor.

Example 10.1

Consider the following ARX and OE models:

$$\text{ARX model : } \hat{y}(k\Delta t) = 0.5y((k-1)\Delta t) + u((k-2)\Delta t) + 0.2 \quad (10.3)$$

$$\text{OE model : } \hat{y}(k\Delta t) = 0.5\hat{y}((k-1)\Delta t) + u((k-2)\Delta t) + 0.2 \quad (10.4)$$

Table 10.1 shows the predicted process outputs (model output) for the given process input and the process output. Table 10.2 is the MATLAB code for Table 10.1. The initial values of the

Table 10.1 Prediction results of an ARX model and an OE model.

Sampling k	Process input $u(k\Delta t)$	Process output $y(k\Delta t)$	Prediction (model output, $\hat{y}(k\Delta t)$)	
			ARX	OE
1	0	0.2300	Not available	0.2300 (initial value)
2	0	0.1600	Not available	0.1600 (initial value)
3	2	0.3000	0.2800	0.2800
4	2	0.3800	0.3500	0.3400
5	2	2.2500	2.3900	2.3700
6	0	3.0500	3.3250	3.3850
7	0	4.2000	3.7250	3.8925
8	1	2.0000	2.3000	2.1463
9	1	1.5700	1.2000	1.2731
10	1	1.6500	1.9850	1.8366

Table 10.2 MATLAB code to simulate Table 10.1.

arx_oe_prediction.m	Command window
clear;	>> arx_oe_prediction
y=[0.23 0.16 0.30 0.38 2.25 3.05 4.20	y_arx =
2.00 1.57 1.65];	0.2300 0.1600 0.2800
u=[0 0 2 2 2 0 0 1 1 1];	0.3500 2.3900 3.3250
y_arx(1)=y(1); y_arx(2)=y(2); %dummy	3.7250 2.3000 1.2000
setting	1.9850
for i=3:10	
y_arx(i)=0.5*y(i-1)+u(i-2)+0.2;	y_oe =
end	0.2300 0.1600 0.2800
y_oe(1)=y(1); y_oe(2)=y(2); %initial	0.3400 2.3700 3.3850
values	3.8925 2.1463 1.2731
for i=3:10	1.8366
y_oe(i)=0.5*y_oe(i-1)+u(i-2)	
+0.2;	
end	
y_arx	
y_oe	
plot(1:10,y_arx,1:10,y_oe,1:10,y);	
legend('ARX','OE','Process Output');	

OE model are chosen as the process outputs. It should be noted that the ARX model can predict only as much as one sampling time because it requires the process output before one sampling time. Meanwhile, the OE model can predict without limit only if the process input is given because it uses the model output. So, the ARX model is a one-step-ahead predictor and the OE model is a multistep-ahead predictor.

10.2 Prediction Error Identification Method for the Autoregressive Exogenous Input Model

The prediction error identification method to identify the ARX model estimates the model parameters by minimizing the prediction error of the ARX model (Ljung, 1987). It solves the following optimization problem to obtain the model parameters from the discrete-sampled-data $y(i\Delta t)$, $i = \hat{d} + n + 1, \dots, N$, and known $u(t)$. The objective function of (10.5) is the norm of the one-step-ahead prediction error.

$$\min_{\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{d}, \hat{B}} \left[V(\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{d}, \hat{B}) = \frac{1}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right]$$

subject to

$$\text{ARX model (10.1)} \tag{10.5}$$

where $y(i\Delta t)$ and $\hat{y}(i\Delta t)$ denote the process output and the model output respectively. It should be noted that, if \hat{d} is given, the model parameters $\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{B}$ can be estimated using the least-squares method. If \hat{d} is not given, then the model parameters $\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{d}, \hat{B}$ should be estimated by solving the nonlinear optimization problem (10.5).

10.2.1 Case 1: Time Delay is Known

Assume that the time delay of $\hat{d}\Delta t$ is known. Then, the optimization problem (10.5) becomes the following form:

$$\min_{\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{B}} \left[V(\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{B}) = \frac{1}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right]$$

subject to

$$\text{ARX model (10.1)} \quad (10.6)$$

Note that all the measurements of the right-hand side of (10.1) are available and (10.1) is a linear form with respect to the model parameters. So, the optimization problem (10.6) can be solved in a straightforward manner by applying the least-squares method to (10.1). For a detailed description of the least-squares method, refer to Chapter 2. The solution of (10.6) is

$$\hat{p} = [\Phi^T \Phi]^{-1} [\Phi^T Y] \quad (10.7)$$

where $\hat{p} = [\hat{a}_1 \ \dots \ \hat{a}_n \ \hat{b}_1 \ \dots \ \hat{b}_n \ \hat{B}]^T$ are the model parameters, and the matrices are as follows:

$$\begin{aligned} \varphi_{1,k} &= -y((k-1)\Delta t), \dots, \varphi_{n,k} = -y((k-n)\Delta t) \\ \varphi_{n+1,k} &= u((k-\hat{d}-1)\Delta t), \dots, \varphi_{n+n,k} = u((k-\hat{d}-n)\Delta t), \varphi_{2n+1,k} = 1 \end{aligned} \quad (10.8)$$

$$Y = \begin{bmatrix} y_{\hat{d}+n} \\ y_{\hat{d}+n+1} \\ \vdots \\ y_N \end{bmatrix}, \quad \Phi = \begin{bmatrix} \varphi_{1,\hat{d}+n} & \varphi_{2,\hat{d}+n} & \cdots & \varphi_{2n+1,\hat{d}+n} \\ \varphi_{1,\hat{d}+n+1} & \varphi_{2,\hat{d}+n+1} & \cdots & \varphi_{2n+1,\hat{d}+n+1} \\ \vdots & \vdots & \ddots & \vdots \\ \varphi_{1,N} & \varphi_{2,N} & \cdots & \varphi_{2n+1,N} \end{bmatrix} \quad (10.9)$$

Y , Φ and \hat{p} are an $(N-n+1) \times 1$ vector, $(N-n+1) \times (2n+1)$ matrix and $(2n+1) \times 1$ vector respectively.

10.2.2 Case 2: Time Delay is Unknown

Assume that the time delay of $\hat{d}\Delta t$ is unknown. Then, the optimization problem (10.5) should be solved. The optimization problem (10.5) can be rewritten (10.10) using the optimal solution of Case 1:

$$\min_{\hat{d}} \left[V(\hat{d}) = \frac{1}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right] \quad \text{subject to} \quad (10.10)$$

(10.7) and ARX model (10.1)

To solve the optimization problem, the interval-halving method can be used because (10.10) is a one-dimensional nonlinear optimization problem. In this case, it should be noted that \hat{d} is an integer. So, the real number chosen by the interval-halving algorithm should be converted to the closest integer number for every iteration and the termination condition should be changed. For the detailed code to reflect this, refer to the Example 10.2.

Example 10.2

The following continuous-time process with time delay is simulated to confirm the identification performance of the prediction error identification method for the discrete-time difference equation model of the ARX model:

$$\frac{d^2 y(t)}{dt^2} + 2 \frac{dy(t)}{dt} + y(t) = u(t - 0.5) \quad (10.11)$$

The process is activated by the P controller, for which the setpoint is changed from 0 to 1 at $t = 1$ and from 1 to 0 at $t = 7$. The sampling time is 0.1. Figure 10.1 shows the activated process input and output.

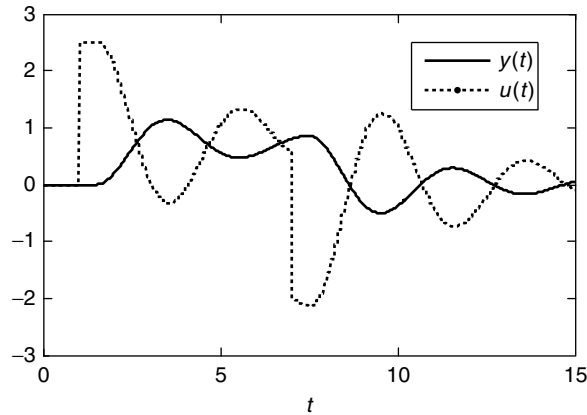


Figure 10.1 Activated process input and output by a P controller.

10.2.1 Autoregressive Exogenous Input Model for the Case that the Time Delay is Known

The time delay of the process is 0.5 as shown in (10.11). So, the number of the sampling time corresponding to the time delay is 5. And, the order of the process is 2 and it is

assumed that the bias term of $\hat{B} = 0$ is known. Then, the discrete-time ARX model should be chosen as (10.12).

$$\hat{y}(k\Delta t) = -\hat{a}_1 y((k-1)\Delta t) - \hat{a}_2 y((k-2)\Delta t) + \hat{b}_1 u((k-6)\Delta t) + \hat{b}_2 u((k-7)\Delta t) \quad (10.12)$$

The model parameters of the ARX model obtained by the prediction error identification method of (10.7) are $\hat{a}_1 = -1.8107394$, $\hat{a}_2 = 0.8197907$, $\hat{b}_1 = 0.0028756$, $\hat{b}_2 = 0.0061637$. The MATLAB code is shown in Table 10.3.

Table 10.3 MATLAB code to estimate the ARX model using the least-squares method.

```

                                PEM_arx1.m
clear; delt=0.1; sub_delt=0.02; tf=15; tref=-delt+sub_delt; ys=0.0;
n=round(tf/sub_delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0 ; 0]; y=0.0; u_data=zeros(1,1001); u=0.0;
k=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.0;
for i=1:n
    t=i*sub_delt; tt(i)=t; ym=y+noise(i); yy(i)=ym; yys(i)=ys;
    if (t>1) ys=1.0; end
    if (t>7) ys=0.0; end
    if (abs(t-(tref+delt))<0.001)
        k=k+1;
        tref=t; u=2.5*(ys-ym); pem_t_data(k)=t; pem_u_data(k)=u;
pem_y_data(k)=ym;
    end
    for j=1:1000 u_data(j)=u_data(j+1); end
    u_data(1001)=u; uu(i)=u;
    [x,y]=g_discrete_arx1_PEM(x,sub_delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(tt,yy,tt,uu); legend('y(t)','u(t)');
% PEM for the ARX model
m=length(pem_u_data);
for k=1:m-7
    Y(k,1)=pem_y_data(k+7);
    phi_1(k,1)=-pem_y_data(k+6);
    phi_2(k,1)=-pem_y_data(k+5);
    phi_3(k,1)=pem_u_data(k+1);
    phi_4(k,1)=pem_u_data(k);
end
PHI=[phi_1 phi_2 phi_3 phi_4];
P=inv(PHI'*PHI)*PHI'*Y;
yy_m=PHI*P;
fprintf('a1=%8.7f a2=%8.7f b1=%8.7f b2=%8.7f \n',P(1),P(2),P(3),P(4));
figure(2); plot(tt,yy,'c',pem_t_data(8:m),yy_m,'k');
legend('process','model');

```

```

                                g_discrete_arx1_PEM.m
function
[next_x,y]=g_discrete_arx1_PEM(x,sub_delt,u,a1,a2,b1,b2,delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);

```

Table 10.3 (Continued)

```

A=[0 -a2; 1 -a1]; B=[b2; b1]; C=[0 1];
delay_k=round(delay/sub_delt);
for i=1:n
    dx=A*x+B*u(1000-delay_k);
    x=x+dx*sub_subdelt;
end
next_x=x; y=C*x;
return

```

Command Window

```

>> PEM_arx1
a1=-1.8107394 a2=0.8197907 b1=0.0028756 b2=0.0061637

```

Example 10.2.1 Autoregressive Exogenous Input Model for the Case that the Time Delay is Unknown

It is assumed that the bias term $\hat{B} = 0$ is known and the time delay of the process is unknown. Then, the discrete-time ARX model should be chosen as

$$\hat{y}(k\Delta t) = -\hat{a}_1 y((k-1)\Delta t) - \hat{a}_2 y((k-2)\Delta t) + \hat{b}_1 u((k-1-\hat{d})\Delta t) + \hat{b}_2 u((k-2-\hat{d})\Delta t) \quad (10.13)$$

The model parameters of the ARX model obtained by the prediction error identification method (10.10) are $\hat{a}_1 = -1.8107394$, $\hat{a}_2 = 0.8197907$, $\hat{b}_1 = 0.0028756$, $\hat{b}_2 = 0.0061637$ and $\hat{d} = 5$. The initial interval for the interval-halving method is chosen as $\hat{d}_{\min} = 0$ and $\hat{d}_{\max} = 10$. The MATLAB code is shown in Table 10.4. It should be noted that the variable \hat{d} is an integer. So, the code rounds off the real number chosen by the interval-halving algorithm to the closest integer number and it terminates when the interval is 1.

Table 10.4 MATLAB code to estimate the ARX model using the interval-halving method and the least-squares method.

```

PEM_arx2.m

clear;
delt=0.1; sub_delt=0.02; tf=15; tref=-delt+sub_delt; ys=0.0;
n=round(tf/sub_delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0; 0]; y=0.0; u_data=zeros(1,1001); u=0.0;
k=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.0;
for i=1:n
    t=i*sub_delt; tt(i)=t; ym=y+noise(i); yy(i)=ym; yys(i)=ys;
    if (t>1) ys=1.0; end
    if (t>7) ys=0.0; end
    if (abs(t-(tref+delt))<0.001)
        k=k+1;
        tref=t; u=2.5*(ys-ym); pem_t_data(k)=t; pem_u_data(k)=u;
    pem_y_data(k)=ym;

```

(continued)

Table 10.4 (Continued)

```

end
for j=1:1000
    u_data(j)=u_data(j+1);
end
u_data(1001)=u; uu(i)=u;
[x,y]=g_discrete_arx2_PEM(x,sub_delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(tt,yy,tt,uu); legend('y(t)','u(t)');
% PEM for the ARX model
% interval-halving method
dmax=10; dmin=0; iter=0;
while (1)
    iter=iter+1;
    interval=(dmax-dmin)/4;
    d1=round(dmin+interval);
    d2=round(dmin+2*interval);
    d3=round(dmin+3*interval);
    [f1,P]=error_arx2_PEM(pem_u_data,pem_y_data,d1);
    [f2,P]=error_arx2_PEM(pem_u_data,pem_y_data,d2);
    [f3,P]=error_arx2_PEM(pem_u_data,pem_y_data,d3);
    if((f1<f2) & (f2<=f3)) dmax=d2; end
    if((f1>=f2) & (f2>f3)) dmin=d2; end
    if((f1>=f2) & (f2<=f3)) dmin=d1; dmax=d3; end
    if(abs(dmax-dmin)==1)
        [f1,Pmin]=error_arx2_PEM(pem_u_data,pem_y_data,dmin);
        [f2,Pmax]=error_arx2_PEM(pem_u_data,pem_y_data,dmax);
        if(f2<=f1) d=dmax; P=Pmax; else d=dmin; P=Pmin; end
        break;
    end
    fprintf(' iter=%2d f1=%8.4e f2=%8.4e f3=%8.4e d1=%2d d2=%2d d3=%2d\n',iter,f1,f2,f3,d1,d2,d3);
end
fprintf(' a1=%8.7f a2=%8.7f b1=%8.7f b2=%8.7f d=%3d\n',P(1),P(2),P(3),P(4),d);

```

error_arx2_PEM.m

```

function [error,P]=error_arx2_PEM(pem_u_data,pem_y_data,d);
m=length(pem_u_data);
for k=1:m-2-d
    Y(k,1)=pem_y_data(k+d+2);
    phi_1(k,1)=-pem_y_data(k+d+1);
    phi_2(k,1)=-pem_y_data(k+d);
    phi_3(k,1)=pem_u_data(k+1);
    phi_4(k,1)=pem_u_data(k);
end
PHI=[phi_1 phi_2 phi_3 phi_4];
P=inv(PHI'*PHI)*PHI'*Y;
yy_m=PHI*P;

```

Table 10.4 (Continued)

```

error=(pem_y_data(d+3:m)-yy_m')*(pem_y_data(d+3:m)-yy_m')'/
(m-d-2);
return

```

```

                                g_discrete_arx2_PEM.m
function
[next_x,y]=g_discrete_arx2_PEM(x,sub_delt,u,a1,a2,b1,b2,delay);
sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
delay_k=round(delay/sub_delt);
for i=1:n
    dx=A*x+B*u(1000-delay_k);
    x=x+dx*sub_subdelt;
end
next_x=x; y=C*x;
return

```

```

                                Command Window
>> PEM_arx2
iter= 1 f1=6.1500e-006 f2=2.5526e-009 f3=1.7149e-005 d1= 3 d2= 5 d3= 8
iter= 2 f1=2.4156e-006 f2=7.2092e-007 f3=7.9410e-006 d1= 4 d2= 6 d3= 7
iter= 3 f1=2.5526e-009 f2=7.2092e-007 f3=7.2092e-007 d1= 5 d2= 6 d3= 6
a1=-1.8107394 a2=0.8197907 b1=0.0028756 b2=0.0061637 d= 5

```

10.3 Prediction Error Identification Method for the Output Error Model

The prediction error identification method to identify the OE model estimates the model parameters by minimizing the prediction error of the OE model (Ljung, 1987). It solves the following optimization problem to obtain the model parameters from the discrete-sampled-data $y(i\Delta t)$, $i = d + n + 1, \dots, N$ and known $u(t)$. The objective function of (10.14) is the norm of the multistep-ahead prediction error:

$$\min_{\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{d}, \hat{B}} \left[V(\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{d}, \hat{B}) = \frac{1}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right]$$

subject to

$$\text{OE model (10.2)} \quad (10.14)$$

where $y(i\Delta t)$ and $\hat{y}(i\Delta t)$ denote the process output and the model output respectively.

10.3.1 Case 1: Time Delay is Known

Assume that the time delay $\hat{d}\Delta t$ is known. Then, the optimization problem (10.14) becomes the following form:

$$\min_{\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{B}} \left[V(\hat{a}_1, \dots, \hat{a}_n, \hat{b}_1, \dots, \hat{b}_n, \hat{B}) = \frac{0.5}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right]$$

subject to

$$\text{OE model 10.2} \quad (10.15)$$

To solve this optimization problem, the following Levenberg–Marquardt method is used, which repeats (10.16) until the parameters converge within tolerance:

$$\hat{p}(j) = \hat{p}(j-1) - \left[\frac{\partial^2 V(\hat{p}(j-1))}{\partial \hat{p}^2} + \alpha I \right]^{-1} \left[\frac{\partial V(\hat{p}(j-1))}{\partial \hat{p}} \right] \quad (10.16)$$

$$\hat{p} = [\hat{a}_1 \quad \hat{a}_2 \quad \dots \quad \hat{a}_n \quad \hat{b}_1 \quad \hat{b}_2 \quad \dots \quad \hat{b}_n \quad \hat{d} \quad \hat{B}]^T \quad (10.17)$$

where j denotes the iteration number and α is a small positive value that can be updated every iteration to compromise between the robustness and the convergence rate. For details, refer to Chapter 2. The initial values in (10.16) are recommended to be chosen as the model parameters obtained by the prediction error identification methods for the ARX model of Section 10.2.

The partial derivatives of the objective function with respect to the adjustable parameters in (10.16) can be calculated by using the numerical derivative or solving the difference equations. Refer to Chapter 2 for the numerical derivative. Refer to the following to understand how to calculate the partial derivatives by solving the difference equations.

From (10.15), (10.18) is derived:

$$\frac{\partial V(\hat{p})}{\partial \hat{p}} = - \frac{1}{N - \hat{d} - n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t)) \frac{\partial \hat{y}(i\Delta t)}{\partial \hat{p}} \quad (10.18)$$

$$\frac{\partial \hat{y}(t)}{\partial \hat{p}} = \left[\frac{\partial \hat{y}(t)}{\partial \hat{a}_1} \quad \dots \quad \frac{\partial \hat{y}(t)}{\partial \hat{a}_n} \quad \frac{\partial \hat{y}(t)}{\partial \hat{b}_1} \quad \dots \quad \frac{\partial \hat{y}(t)}{\partial \hat{b}_n} \quad \frac{\partial \hat{y}(t)}{\partial \hat{B}} \right]^T \quad (10.19)$$

From (10.2), the partial derivatives (10.20)–(10.24) are obtained for \hat{a}_1 , \hat{a}_2 , \hat{b}_1 , \hat{b}_2 and \hat{B} . Also, the other partial derivatives can be derived in a similar way. Then, the partial derivatives in (10.18) can be calculated by solving the difference equations.

$$\frac{\partial \hat{y}(k\Delta t)}{\partial \hat{a}_1} = -\hat{y}((k-1)\Delta t) - \hat{a}_1 \frac{\partial \hat{y}((k-1)\Delta t)}{\partial \hat{a}_1} - \hat{a}_2 \frac{\partial \hat{y}((k-2)\Delta t)}{\partial \hat{a}_1} - \dots - \hat{a}_n \frac{\partial \hat{y}((k-n)\Delta t)}{\partial \hat{a}_1} \quad (10.20)$$

$$\frac{\partial \hat{y}(k\Delta t)}{\partial \hat{a}_2} = -\hat{y}((k-2)\Delta t) - \hat{a}_1 \frac{\partial \hat{y}((k-1)\Delta t)}{\partial \hat{a}_2} - \hat{a}_2 \frac{\partial \hat{y}((k-2)\Delta t)}{\partial \hat{a}_2} - \dots - \hat{a}_n \frac{\partial \hat{y}((k-n)\Delta t)}{\partial \hat{a}_2} \quad (10.21)$$

$$\frac{\partial \hat{y}(k\Delta t)}{\partial \hat{b}_1} = -\hat{a}_1 \frac{\partial \hat{y}((k-1)\Delta t)}{\partial \hat{b}_1} - \hat{a}_2 \frac{\partial \hat{y}((k-2)\Delta t)}{\partial \hat{b}_1} - \dots - \hat{a}_n \frac{\partial \hat{y}((k-n)\Delta t)}{\partial \hat{b}_1} + u((k-1-\hat{d})\Delta t) \quad (10.22)$$

$$\frac{\partial \hat{y}(k\Delta t)}{\partial \hat{b}_2} = -\hat{a}_1 \frac{\partial \hat{y}((k-1)\Delta t)}{\partial \hat{b}_2} - \hat{a}_2 \frac{\partial \hat{y}((k-2)\Delta t)}{\partial \hat{b}_2} - \dots - \hat{a}_n \frac{\partial \hat{y}((k-n)\Delta t)}{\partial \hat{b}_2} + u((k-2-\hat{d})\Delta t) \quad (10.23)$$

$$\frac{\partial \hat{y}(k\Delta t)}{\partial \hat{B}} = -\hat{a}_1 \frac{\partial \hat{y}((k-1)\Delta t)}{\partial \hat{B}} - \hat{a}_2 \frac{\partial \hat{y}((k-2)\Delta t)}{\partial \hat{B}} - \dots - \hat{a}_n \frac{\partial \hat{y}((k-n)\Delta t)}{\partial \hat{B}} + 1 \quad (10.24)$$

The initial values of all the partial derivatives are zero because the parameter of \hat{p} does not affect the initial values of the partial derivatives.

The second derivative of (10.16) is

$$\begin{aligned} \frac{\partial^2 V(\hat{p})}{\partial \hat{p}^2} = & -\frac{1}{N-\hat{d}-n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t)) \frac{\partial^2 \hat{y}(i\Delta t)}{\partial \hat{p}^2} \\ & + \frac{1}{N-\hat{d}-n} \sum_{i=\hat{d}+n+1}^N \left[\frac{\partial \hat{y}(i\Delta t)}{\partial \hat{p}} \right] \left[\frac{\partial \hat{y}(i\Delta t)}{\partial \hat{p}} \right]^T \end{aligned} \quad (10.25)$$

When the solution is close to actuality, the first term of the right-hand side in (10.25) can be neglected:

$$\frac{\partial^2 V(\hat{p})}{\partial \hat{p}^2} \approx \frac{1}{N-\hat{d}-n} \sum_{i=\hat{d}+n+1}^N \left[\frac{\partial \hat{y}(i\Delta t)}{\partial \hat{p}} \right] \left[\frac{\partial \hat{y}(i\Delta t)}{\partial \hat{p}} \right]^T \quad (10.26)$$

In summary, all the partial derivatives in (10.18) and (10.26) are calculated for a given $\hat{p}(j-1)$ by selecting them at every sampling while continuously solving the difference equations such as (10.2) and (10.20)–(10.24) simultaneously. Next, it is straightforward to calculate the updated parameters $\hat{p}(j)$ from (10.16). Repeat this procedure until the parameters converge.

10.3.2 Case 2: Time Delay is Unknown

Assume that the time delay $\hat{d}\Delta t$ is unknown. Then, the optimization problem (10.14) should be solved. The optimization problem (10.14) can be rewritten (10.27) using the optimal solution of Case 1:

$$\min_{\hat{d}} \left[V(\hat{d}) = \frac{1}{N-\hat{d}-n} \sum_{i=\hat{d}+n+1}^N (y(i\Delta t) - \hat{y}(i\Delta t))^2 \right] \quad \text{subject to} \quad \text{solutions of (10.15)} \quad (10.27)$$

To solve the optimization problem, the interval-halving method can be used because (10.27) is a one-dimensional nonlinear optimization problem. In this case, it should be noted that \hat{d} is an integer. So, the real number chosen by the interval-halving algorithm should be converted to the closest integer number for every iteration and the termination condition should be changed. For the detailed code to reflect this, refer to the Example 10.3.

Example 10.3

Estimate the OE model for the activated process input and the process output in Example 10.2.

Solution Two cases are considered. The first case assumes that the time delay is known. The second case assumes that the time delay is unknown.

Example 10.3.1 Output Error Model for the Case that the Time Delay is Known

It is assumed that the time delay of the process is 0.5, as shown in (10.11). So, the number of the sampling time corresponding to the time delay is 5. Also, the order of the process is 2 and it is assumed that the bias term $\hat{B} = 0$ is known. Then, the discrete-time OE model should be chosen as

$$\hat{y}(k\Delta t) = -\hat{a}_1\hat{y}((k-1)\Delta t) - \hat{a}_2\hat{y}((k-2)\Delta t) + \hat{b}_1u((k-6)\Delta t) + \hat{b}_2u((k-7)\Delta t) \quad (10.28)$$

The model parameters of the OE model obtained by the prediction error identification method of (10.15) are $\hat{a}_1 = -1.8142$, $\hat{a}_2 = 0.823\,05$, $\hat{b}_1 = 0.004\,443$ and $\hat{b}_2 = 0.004\,425\,1$. The initial values for the Levenberg–Marquardt method are chosen as $\hat{a}_1 = -1.5$, $\hat{a}_2 = 0.60$, $\hat{b}_1 = 0.002$ and $\hat{b}_2 = 0.003$. The MATLAB code is shown in Table 10.5.

Example 10.3.2 Output Error Model for the Case that the Time Delay is Unknown

It is assumed that the time delay of the process is unknown and the bias term $\hat{B} = 0$ is known. Then, the discrete-time OE model should be chosen as

$$\hat{y}(k\Delta t) = -\hat{a}_1\hat{y}((k-1)\Delta t) - \hat{a}_2\hat{y}((k-2)\Delta t) + \hat{b}_1u((k-1-\hat{d})\Delta t) + \hat{b}_2u((k-2-\hat{d})\Delta t) \quad (10.29)$$

The model parameters of the OE model obtained by the prediction error identification method of (10.27) are $\hat{a}_1 = -1.8142$, $\hat{a}_2 = 0.823\,05$, $\hat{b}_1 = 0.004\,443$, $\hat{b}_2 = 0.004\,425\,1$ and $\hat{d} = 5$. The initial values for the Levenberg–Marquardt method are chosen as $\hat{a}_1 = -1.5$, $\hat{a}_2 = 0.60$, $\hat{b}_1 = 0.002$ and $\hat{b}_2 = 0.003$. And the initial interval for the interval-halving

Table 10.5 MATLAB code to estimate the OE model using the Levenberg–Marquardt method.

```

                                PEM_oe1.m

clear;
delt=0.1; sub_delt=0.01; tf=15; tref=-delt+sub_delt; ys=0.0;
n=round(tf/sub_delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0 ; 0]; y=0.0; u_data=zeros(1,1001); u=0.0;
k=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.0;
for i=1:n
    t=i*sub_delt; tt(i)=t; ym=y+noise(i); yy(i)=ym; yys(i)=ys;
    if (t>1) ys=1.0; end
    if (t>7) ys=0.0; end
    if (abs(t-(tref+delt))<0.001)
        k=k+1;
        tref=t; u=2.5*(ys-ym); pem_t_data(k)=t; pem_u_data(k)=u;
pem_y_data(k)=ym;
    end
    for j=1:1000
        u_data(j)=u_data(j+1);
    end
    u_data(1001)=u; uu(i)=u;
    [x,y]=g_discrete_oe1_PEM(x,sub_delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(tt,yy,tt,uu); legend('y(t)','u(t)');
% PEM for the OE model
a1=-1.5; a2=0.6; b1=0.002; b2=0.003; %initial values for the LV method
[P,E]=lv_oe1(pem_u_data,pem_y_data,a1,a2,b1,b2,5);
fprintf('a1=%8.4e a2=%8.4e b1=%8.4e b2=%8.4e d=%2d E=%8.4e\n',P(1),P(2),P(3),P(4),5,E);

```

```

                                lv_oe1.m

function [P,E_new]=lv_oe1(pem_u_data,pem_y_data,a1,a2,b1,b2,d)
delta=0.00001; %interval for the numerical derivatives
alpha=1.0; index_update=1; iter=0;
Pb=[a1 a2 b1 b2 ]';
E=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2,b1,b2,d);
%fprintf('iter=%2d a1=%8.4e a2=%8.4e b1=%8.4e b2=%8.4e d=%2d\n',iter,Pb(1),Pb(2),Pb(3),Pb(4),d,E);
E=%8.4e\n', iter, Pb(1), Pb(2), Pb(3), Pb(4), d, E);
while(1)
    if (index_update==1)
        a1=Pb(1); a2=Pb(2); b1=Pb(3); b2=Pb(4);
        E=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2,b1,b2,d); %object
function
        E_a1=object_PEM_oe1(pem_u_data,pem_y_data,a1+delta,a2,b1,b2,d);
        E_a2=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2+delta,b1,b2,d);
        E_b1=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2,b1+delta,b2,d);
        E_b2=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2,b1,b2+delta,d);
        E_a1_a1=object_PEM_oe1(pem_u_data,pem_y_data,a1+2*delta,a2,b1,b2,d);
        E_a2_a2=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2+2*delta,b1,b2,d);
        E_b1_b1=object_PEM_oe1(pem_u_data,pem_y_data,a1,a2,b1+2*delta,b2,d);

```

(continued)

Table 10.5 (Continued)

```

E_b2_b2=object_PEM_oel(pem_u_data,pem_y_data,a1,a2,b1,b2+2*delta,d);
E_a1_a2=object_PEM_oel(pem_u_data,pem_y_data,a1+delta,
a2+delta,b1,b2,d);
E_a1_b1=object_PEM_oel(pem_u_data,pem_y_data,a1+delta,a2,
b1+delta,b2,d);
E_a1_b2=object_PEM_oel(pem_u_data,pem_y_data,a1+delta,a2,b1,
b2+delta,d);
E_a2_b1=object_PEM_oel(pem_u_data,pem_y_data,a1,a2+delta,
b1+delta,b2,d);
E_a2_b2=object_PEM_oel(pem_u_data,pem_y_data,a1,a2+delta,b1,
b2+delta,d);
E_b1_b2=object_PEM_oel(pem_u_data,pem_y_data,a1,a2,b1+delta,
b2+delta,d);

dV(1,1)=(E_a1-E)/delta;%dV/dp1
dV(2,1)=(E_a2-E)/delta;%dV/dp2
dV(3,1)=(E_b1-E)/delta;%dV/dp3
dV(4,1)=(E_b2-E)/delta;%dV/dp4

ddV(1,1)=(E_a1_a1-2*E_a1+E)/delta^2;%d^2V/dp1^2
ddV(2,2)=(E_a2_a2-2*E_a2+E)/delta^2;
ddV(3,3)=(E_b1_b1-2*E_b1+E)/delta^2;
ddV(4,4)=(E_b2_b2-2*E_b2+E)/delta^2;

ddV(1,2)=(E_a1_a2-E_a1-E_a2+E)/delta^2;%d^2V/dp1dp2
ddV(1,3)=(E_a1_b1-E_a1-E_b1+E)/delta^2;
ddV(1,4)=(E_a1_b2-E_a1-E_b2+E)/delta^2;

ddV(2,3)=(E_a2_b1-E_a2-E_b1+E)/delta^2;
ddV(2,4)=(E_a2_b2-E_a2-E_b2+E)/delta^2;

ddV(3,4)=(E_b1_b2-E_b1-E_b2+E)/delta^2;

ddV(2,1)=ddV(1,2); ddV(3,1)=ddV(1,3); ddV(4,1)=ddV(1,4);
ddV(3,2)=ddV(2,3); ddV(4,2)=ddV(2,4);
ddV(4,3)=ddV(3,4);
end
P=Pb-inv(ddV+alpha*eye(4))*dV;
a1=P(1); a2=P(2); b1=P(3); b2=P(4);
[E_new yy_m]=object_PEM_oel(pem_u_data,pem_y_data,a1,a2,b1,b2,d);
if (E_new<E)
    index_update=1;
    alpha=alpha/2.0; Pb=P;
    iter=iter+1;
    fprintf(' iter=%2d a1=%5.3e a2=%5.3e b1=%5.3e b2=%5.3e d=%2d
E=%5.3e\n',iter,P(1),P(2),P(3),P(4),d,E);
else
    index_update=0;

```

Table 10.5 (Continued)

```

    alpha=alpha*1.5;
end
if (iter==25 | alpha>10.0^10) break; end
end

```

```

                                object_PEM_oel.m
function [V yy_m]=object_PEM_oel(uu,yy,a1,a2,b1,b2,d)
    n=length(uu);
    for i=1:(d+2) yy_m(i)=yy(i); end %initial values
    for i=(d+3):n
        yy_m(i)=-a1*yy_m(i-1)-a2*yy_m(i-2)+b1*uu(i-1-d)+b2*uu(i-2-d);
    end
    V=(yy-yy_m)*(yy-yy_m)/(n-d-2);
return

```

```

                                g_discrete_oel_PEM.m
function [next_x,y]=g_discrete_oel_PEM(x,sub_delt,u,a1,a2,b1,b2,
delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return

```

```

                                object_PEM_oel.m
function [V yy_m]=object_PEM_oel(uu,yy,a1,a2,b1,b2,d)
    n=length(uu);
    for i=1:(d+2) yy_m(i)=yy(i); end %initial values
    for i=(d+3):n
        yy_m(i)=-a1*yy_m(i-1)-a2*yy_m(i-2)+b1*uu(i-1-d)+b2*uu(i-2-d);
    end
    V=(yy-yy_m)*(yy-yy_m)/(n-d-2);
return

```

```

                                g_discrete_oel_PEM.m
function [next_x,y]=g_discrete_oel_PEM(x,sub_delt,u,a1,a2,b1,b2,
delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return

```

(continued)

Table 10.5 (Continued)

Command Window

```
>> PEM_oe1
iter= 1 a1=-1.497e+000 a2=6.663e-001 b1=-7.420e-002 b2=7.653e-002 d= 5
E=2.583e-001
iter= 2 a1=-1.526e+000 a2=6.341e-001 b1=-6.690e-002 b2=9.757e-002 d= 5
E=2.449e-001 iter=19 a1=-1.814e+000 a2=8.231e-001 b1=4.457e-003
b2=4.408e-003 d= 5 E=1.179e-005
iter=20 a1=-1.814e+000 a2=8.230e-001 b1=4.443e-003 b2=4.425e-003 d= 5
E=6.704e-006 a1=-1.8142e+000 a2=8.2305e-001 b1=4.4433e-003 b2=4.4251e-
003 d= 5 E=6.5084e-006
```

method is $\hat{d}_{\min} = 0$ and $\hat{d}_{\max} = 10$. The MATLAB code is shown in Table 10.6. It should be noted that the variable \hat{d} is an integer. So, the code rounds off the real number chosen by the interval-halving algorithm to the closest integer number and it terminates when the interval is 1.

Table 10.6 MATLAB code to estimate the OE model using the interval-halving method and the Levenberg–Marquardt method.

PEM_oe2.m

```
clear;
delt=0.1; sub_delt=0.01; tf=15; tref=-delt+sub_delt; ys=0.0; n=round
(tf/sub_delt);
a1=2.0; a2=1.0; b1=0.0; b2=1.0; delay=0.5; %process
C=[0 1]; x=[0 ; 0]; y=0.0; u_data=zeros(1,1001); u=0.0;
k=0; rand('seed',0); noise=(rand(1,n)-0.5)*0.0;
for i=1:n
    t=i*sub_delt; tt(i)=t; ym=y+noise(i); yy(i)=ym; yys(i)=ys;
    if (t>1) ys=1.0; end
    if (t>7) ys=0.0; end
    if (abs(t-(tref+delt))<0.001)
        k=k+1;
        tref=t; u=2.5*(ys-ym); pem_t_data(k)=t; pem_u_data(k)=u;
pem_y_data(k)=ym;
    end
    for j=1:1000
        u_data(j)=u_data(j+1);
    end
    u_data(1001)=u; uu(i)=u;
    [x,y]=g_discrete_oe2_PEM(x,sub_delt,u_data,a1,a2,b1,b2,delay);
end
figure(1); plot(tt,yy,tt,uu); legend('y(t)','u(t)');
% PEM for the OE model
% interval-halving method
dmax=10; dmin=0; iter=0;
a1=-1.5; a2=0.6; b1=0.002; b2=0.003; %initial values for the LV method
```

Table 10.6 (Continued)

```

while (1)
    iter=iter+1;
    interval=(dmax-dmin)/4;
    d1=round(dmin+interval);
    d2=round(dmin+2*interval);
    d3=round(dmin+3*interval);
    [P,f1]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d1);
    [P,f2]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d2);
    [P,f3]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d3);
    if((f1<f2) & (f2<=f3)) dmax=d2; end
    if((f1>=f2) & (f2>f3)) dmin=d2; end
    if((f1>=f2) & (f2<=f3)) dmin=d1; dmax=d3; end
    if(abs(dmax-dmin)==1)
        [Pmin,f1]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,dmin);
        [Pmax,f2]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,dmax);
        if(f2<=f1) d=dmax; P=Pmax; else d=dmin; P=Pmin; end
        break;
    end
    fprintf(' iter=%2d f1=%8.4e f2=%8.4e f3=%8.4e d1=%2d d2=%2d d3=%2d
\n',iter,f1,f2,f3,d1,d2,d3);
end
fprintf(' a1=%8.7f a2=%8.7f b1=%8.7f b2=%8.7f d=%2d
\n',P(1),P(2),P(3),P(4),d);

```

```

lv_oe2.m
function [P,E_new]=lv_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d)
delta=0.00001; %interval for the numerical derivatives
alpha=1.0; index_update=1; iter=0;
Pb=[a1 a2 b1 b2]';
E=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d);
%fprintf(' iter=%2d a1=%8.4e a2=%8.4e b1=%8.4e b2=%8.4e d=%2d
E=%8.4e\n',iter,Pb(1),Pb(2),Pb(3),Pb(4),d,E);

while(1)
    if(index_update==1)
        a1=Pb(1); a2=Pb(2); b1=Pb(3); b2=Pb(4);
        E=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d); %object
function
        E_a1=object_PEM_oe2(pem_u_data,pem_y_data,a1+delta,a2,b1,b2,d);
        E_a2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2+delta,b1,b2,d);
        E_b1=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1+delta,b2,d);
        E_b2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2+delta,d);

        E_a1_a1=object_PEM_oe2(pem_u_data,pem_y_data,a1+2*delta,a2,b1,b2,d);
        E_a2_a2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2+2*delta,b1,b2,d);
        E_b1_b1=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1+2*delta,b2,d);
        E_b2_b2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2+2*delta,d);
        E_a1_a2=object_PEM_oe2(pem_u_data,pem_y_data,a1+delta,a2+delta,b1,
b2,d);

```

(continued)

Table 10.6 (Continued)

```

E_a1_b1=object_PEM_oe2(pem_u_data,pem_y_data,a1+delta,a2,b1+delta,
b2,d);
E_a1_b2=object_PEM_oe2(pem_u_data,pem_y_data,a1+delta,a2,b1,b2+
delta,d);
E_a2_b1=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2+delta,b1+delta,
b2,d);
E_a2_b2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2+delta,b1,b2+
delta,d);
E_b1_b2=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1+delta,b2+
delta,d);

dV(1,1)=(E_a1-E)/delta;%dV/dp1
dV(2,1)=(E_a2-E)/delta;%dV/dp2
dV(3,1)=(E_b1-E)/delta;%dV/dp3
dV(4,1)=(E_b2-E)/delta;%dV/dp4

ddV(1,1)=(E_a1_a1-2*E_a1+E)/delta^2;%d^2V/dp1^2
ddV(2,2)=(E_a2_a2-2*E_a2+E)/delta^2;
ddV(3,3)=(E_b1_b1-2*E_b1+E)/delta^2;
ddV(4,4)=(E_b2_b2-2*E_b2+E)/delta^2;
ddV(1,2)=(E_a1_a2-E_a1-E_a2+E)/delta^2;%d^2V/dp1dp2
ddV(1,3)=(E_a1_b1-E_a1-E_b1+E)/delta^2;
ddV(1,4)=(E_a1_b2-E_a1-E_b2+E)/delta^2;

ddV(2,3)=(E_a2_b1-E_a2-E_b1+E)/delta^2;
ddV(2,4)=(E_a2_b2-E_a2-E_b2+E)/delta^2;

ddV(3,4)=(E_b1_b2-E_b1-E_b2+E)/delta^2;

ddV(2,1)=ddV(1,2); ddV(3,1)=ddV(1,3); ddV(4,1)=ddV(1,4);
ddV(3,2)=ddV(2,3); ddV(4,2)=ddV(2,4);
ddV(4,3)=ddV(3,4);
end
P=Pb-inv(ddV+alpha*eye(4))*dV;
a1=P(1); a2=P(2); b1=P(3); b2=P(4);
[E_new yy_m]=object_PEM_oe2(pem_u_data,pem_y_data,a1,a2,b1,b2,d);
if (E_new<E)
    index_update=1;
    alpha=alpha/2.0; Pb=P;
    iter=iter+1;
% fprintf('iter=%2d a1=%5.3e a2=%5.3e b1=%5.3e b2=%5.3e d=%2d
E=%5.3e\n',iter,P(1),P(2),P(3),P(4),d,E);
else
    index_update=0;
    alpha=alpha*1.5;
end
if (iter==25 | alpha>10.0^10) break; end
end

```


Table 10.6 (Continued)

```

                                object_PEM_oe2.m
function [V yy_m]=object_PEM_oe2(uu,yy,a1,a2,b1,b2,d)
    n=length(uu);
    for i=1:(d+2) yy_m(i)=yy(i); end %initial values
    for i=(d+3):n
        yy_m(i)=-a1*yy_m(i-1)-a2*yy_m(i-2)+b1*uu(i-1-d)+b2*uu(i-2-d);
    end
    V=(yy-yy_m)*(yy-yy_m)'/(n-d-2);
return

```

```

                                g_discrete_oe2_PEM.m
function [next_x,y]=g_discrete_oe2_PEM(x,
sub_delt,u,a1,a2,b1,b2,delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return

```

```

                                g_discrete_oe2_PEM.m
function [next_x,y]=g_discrete_oe2_PEM(x,sub_delt,u,a1,a2,b1,b2,
delay);
    sub_subdelt=0.005; n=round(sub_delt/sub_subdelt);
    A=[0 -a2 ; 1 -a1]; B=[b2 ; b1]; C=[0 1];
    delay_k=round(delay/sub_delt);
    for i=1:n
        dx=A*x+B*u(1000-delay_k);
        x=x+dx*sub_subdelt;
    end
    next_x=x; y=C*x;
return

```

Command Window

```

>> PEM_oe2
iter= 1 f1=3.4413e-005 f2=6.5084e-006 f3=2.7655e-004 d1= 3 d2= 5 d3= 8
iter= 2 f1=7.4870e-006 f2=1.2640e-005 f3=5.0833e-005 d1= 4 d2= 6 d3= 7
a1=-1.8141750 a2=0.8230493 b1=0.0044433 b2=0.0044251 d= 5

```

10.4 Concluding Remarks

The ARX model predicts a one-step-ahead process output and the OE model predicts a multistep-ahead process output. So, the prediction error identification method for the ARX model estimates the model parameters by minimizing the one-step-ahead prediction errors. Meanwhile, the prediction error identification method for the OE model estimates the model

parameters by minimizing the multistep-ahead prediction errors. In most cases, the process model should have a good capability of multistep-ahead prediction. So, the OE model is usually preferred to the ARX model. But the prediction error identification method for the OE model requires solving a complicated multidimensional nonlinear optimization problem to obtain the model parameters, whereas the prediction error identification method for the ARX model obtains the model parameters in a very simple way using the least-squares method. The initial values for the nonlinear optimization method in the prediction error identification method for the OE model are recommended to be chosen as the model parameters obtained by the prediction error identification method for the ARX model.

Problems

- 10.1 Activate the following process using the biased-relay with a sampling time of 0.2 and estimate the ARX model with a known time delay. Compare the model output and the process output. In this case, calculate the model output using the OE model for which the coefficients are those of the ARX model.

$$\frac{d^3y(t)}{dt^3} + 3\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = u(t - 0.2)$$

- 10.2 Estimate the ARX model with an unknown time delay for the activate process data of Problem 10.1 and compare the model output and the process output. In this case, assume that the ARX model is an SOPTD model. Also, calculate the model output using the OE model for which the coefficients are those of the ARX model.
- 10.3 Estimate the OE model with a known time delay for the activated process data of Problem 10.1 and compare the model output and the process output. In this case, determine the initial estimates using the prediction error method for an ARX model.
- 10.4 Estimate the OE model with the unknown time delay for the activated process data of Problem 10.1 and compare the model output and the process output. In this case, determine the initial estimates using the prediction error method for an ARX model.
- 10.5 Activate the virtual process of Process 3 (refer to the Appendix for details) using a biased-relay and estimate the ARX model with an unknown time delay. Compare the model output and the process output. In this case, calculate the model output using an OE model for which the coefficients are those of the ARX model.
- 10.6 Activate the virtual process of Process 3 (refer to the Appendix for details) using a biased-relay and estimate the OE model with an unknown time delay. Compare the model output and the process output.

Reference

Ljung, L. (1987) *System Identification*, Prentice-Hall, Englewood Cliffs, NJ.

11

Model Conversion from Discrete-Time to Continuous-Time Linear Models

11.1 Transfer Function of Discrete-Time Processes

In Part One, the Laplace transform was used to derive the transfer function of a continuous-time process. For a discrete-time process, the z -transform is used. Consider the following discrete-time process:

$$y(k\Delta t) = -a_1y((k-1)\Delta t) - a_2y((k-2)\Delta t) - \cdots - a_ny((k-n)\Delta t) + b_1u((k-1-d)\Delta t) + b_2u((k-2-d)\Delta t) + \cdots + b_nu((k-n-d)\Delta t) \quad (11.1)$$

The z -transform of $y(k\Delta t)$ is defined as

$$y(z) = Z\{y(k\Delta t)\} = \sum_{k=0}^{\infty} y(k\Delta t)z^{-k} = y(0\Delta t) + y(1\Delta t)z^{-1} + y(2\Delta t)z^{-2} + \cdots \quad (11.2)$$

One of the notable properties of the z -transform is $y(z)z^{-1} = Z\{y(k-1)\Delta t\}$ if $y(k\Delta t) = 0$, $k < 0$. The property is derived straightforwardly by comparing (11.2) with (11.3):

$$Z\{y((k-1)\Delta t)\} = y(-1\Delta t) + y(0\Delta t)z^{-1} + y(1\Delta t)z^{-2} + y(2\Delta t)z^{-3} + \cdots \quad (11.3)$$

From (11.2) and (11.3), it is clear that $y(z)z^{-1} = Z\{y(k-1)\Delta t\}$ if $y(k\Delta t) = 0$, $k < 0$. Equivalently, $y(z)z^{-d} = Z\{y(k-d)\Delta t\}$ if $y(k\Delta t) = 0$, $k < 0$. Then, (11.4) is obtained by applying the z -transform to (11.1):

$$y(z) = -a_1y(z)z^{-1} - a_2y(z)z^{-2} - \cdots - a_ny(z)z^{-n} + b_1u(z)z^{-1-d} + b_2u(z)z^{-2-d} + \cdots + b_nu(z)z^{-n-d} \quad (11.4)$$

Rearranging (11.4), the following transfer function for the discrete-time process is obtained:

$$G(z) = \frac{y(z)}{u(z)} = \frac{b_1 z^{-1} + b_2 z^{-2} + \cdots + b_n z^{-n}}{1 + a_1 z^{-1} + \cdots + a_n z^{-n}} z^{-d} \quad (11.5)$$

or

$$G(z) = \frac{y(z)}{u(z)} = \frac{b_1 z^{n-1} + b_2 z^{n-2} + \cdots + b_n}{z^n + a_1 z^{n-1} + \cdots + a_n} z^{-d} \quad (11.6)$$

11.2 Frequency Responses of Discrete-Time Processes and Model Conversion

The frequency response of the process can be obtained directly from the transfer function without simulation or plant test. Assume that $G(z)$ of the transfer function of the process is available. Then, the amplitude ratio and the phase angle at the frequency ω can be estimated by setting $z = \exp(i\omega\Delta t)$ as shown in (11.7) and (11.8):

$$\text{AR}(\omega) = |G(\exp(i\omega\Delta t))| = \sqrt{\text{Re}(G(\exp(i\omega\Delta t)))^2 + \text{Im}(G(\exp(i\omega\Delta t))))^2} \quad (11.7)$$

$$\phi(\omega) = \angle G(\exp(i\omega\Delta t)) = \arctan \text{ of } \frac{\text{Im}(G(\exp(i\omega\Delta t)))}{\text{Re}(G(\exp(i\omega\Delta t)))} \quad \text{and} \quad \text{Im}(G(\exp(i\omega\Delta t))) \quad (11.8)$$

If the sampling time is small, then the frequency responses in the discrete-time model are almost the same as those of the continuous-time model. Then, the frequency responses of the continuous-time transfer function can be estimated approximately by estimating the frequency responses of (11.7) and (11.8). Then, the discrete-time model can be converted to the corresponding continuous-time model by applying the model reduction techniques mentioned in Chapter 5.

Assume that there is a discrete-time transfer function $G(z)$ and it is required to convert it to a continuous-time SOPTD model:

$$G(z) \cong \frac{k \exp(-\theta s)}{\tau^2 s^2 + 2\tau\xi s + 1} \quad (11.9)$$

The frequency response of the discrete-time model is used. The model reduction method first estimates the gain of the continuous-time model to fit the zero frequency-response data as follows:

$$k = G(z)|_{z=\exp(i0\Delta t)=1} \quad (11.10)$$

and it estimates τ and ξ to satisfy the equality of (11.11) by solving (11.12) using the least-squares method. Equation (11.12) is derived from (11.11) in a straightforward manner:

$$|G(\exp(i\omega_i\Delta t))| \approx \left| \frac{k \exp(-i\theta\omega)}{1 - \tau^2\omega^2 + i2\tau\xi\omega} \right| = \frac{k}{\sqrt{(1 - \tau^2\omega^2)^2 + (2\tau\xi\omega)^2}} \quad (11.11)$$

$$\tau^4 |G(\exp(i\omega_i \Delta t))|^2 \omega_j^4 + (4\tau^2 \xi^2 - 2\tau^2) |G(\exp(i\omega_i \Delta t))|^2 \omega_j^2 = k^2 - |G(\exp(i\omega_i \Delta t))|^2 \quad (11.12)$$

$$0 < \omega_1 < \omega_2 < \dots < \omega_j < \dots < \omega_n \quad (11.13)$$

where it is recommended to choose ω_n as the ultimate frequency ω_u of the discrete-time transfer function. If ω_u is not available, then it should be chosen as the closest one to ω_u . The model reduction method finally estimates the time delay of the continuous-time SOPTD model from the phase-angle equation (11.14) with respect to ω_m . Equation (11.15) is obtained directly from (11.14):

$$\phi(\omega_m) = -\theta\omega_m + \arctan 2(-2\xi\omega_m\tau, 1 - \omega_m^2\tau^2) \quad (11.14)$$

$$\theta = \frac{-\phi(\omega_m) + \arctan 2(-2\xi\tau\omega_m, 1 - \omega_m^2\tau^2)}{\omega_m} \quad (11.15)$$

where $\phi(\omega_m)$ is the phase angle of $G(z)$ at ω_m . ω_m should be $\omega_m \leq \omega_u$. It is recommended to choose ω_m as a frequency close to the ultimate frequency ω_u of the process. If $\omega_m = \omega_u$ is chosen, then (11.14) and (11.15) become the following equations:

$$-\pi = -\theta\omega_u + \arctan 2(-2\xi\omega_u\tau, 1 - \omega_u^2\tau^2) \quad (11.16)$$

$$\theta = \frac{\pi + \arctan 2(-2\xi\tau\omega_u, 1 - \omega_u^2\tau^2)}{\omega_u} \quad (11.17)$$

In summary, the continuous-time SOPTD model (11.9) can be estimated from (11.10), (11.12), and (11.15).

Similarly, (11.18)–(11.20) can be derived to convert the discrete-time model to the continuous-time FOPTD model:

$$k = G(z)|_{z=\exp(i0\Delta t)=1} \quad (11.18)$$

$$\tau = \frac{\sqrt{k^2 - |G(\exp(i\omega_m \Delta t))|^2}}{|G(\exp(i\omega_m \Delta t))|\omega_m} \quad (11.19)$$

$$\theta = \frac{-\phi(\omega_m) + \arctan(-\tau\omega_m)}{\omega_m} \quad (11.20)$$

Example 11.1

Obtain the transfer function of the discrete-time OE model obtained in Example 10.3 in Chapter 10.

Solution The discrete-time model obtained in Chapter 10 is

$$\hat{y}(k\Delta t) = -\hat{a}_1\hat{y}((k-1)\Delta t) - \hat{a}_2\hat{y}((k-2)\Delta t) + \hat{b}_1u((k-1-\hat{d})\Delta t) + \hat{b}_2u((k-2-\hat{d})\Delta t) \quad (11.21)$$

Table 11.1 MATLAB code for the model conversion in Example 11.2.

Conversion_d2c_ex2.m	g_discrete_ex2.m
<pre> clear; w=0.0; delta_w=0.05; delt=0.1; %sampling time while(1) % search boundary in which wu exists w=w+delta_w; g=g_discrete_ex2(w,delt); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_discrete_ex2(w1,delt); g=g_discrete_ex2(w,delt); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found k=abs(g_discrete_ex2(0.0,delt)); for j=1:10 % least square method w=(j-1)*wu/9.0; G(j)=g_discrete_ex2(w,delt); y(j,1)=k-(abs(G(j))^2); phi_1(j,1)=(abs(G(j))^2)*w^4; phi_2(j,1)=(abs(G(j))^2)*w^2; end % P_hat: solution of the least square method PHI=[phi_1 phi_2]; Y=y; P_hat=inv(PHI'*PHI)*PHI'*Y; tau=P_hat(1)^(1.0/4.0); xi=((P_hat(2)+2*tau^2)/ (4*tau^2))^0.5; theta=(pi+atan2(-2*xi*tau*wu,1- wu^2*tau^2))/wu; % tau: time constant, xi: damping factor, theta: time delay fprintf('k=%5.3f tau=%5.3f \n', k,tau); fprintf('xi=%5.3f theta=%5.3f \n',xi,theta); </pre>	<pre> function [G]=g_discrete_ex2(w,delt) z=exp(i*w*delt); G=z^(-5)*(0.004443*z^(-1) +0.0044251*z^(-2))/(1- 1.8142*z^(-1)+0.82305*z^(-2)); end </pre>
	<p>Command Window</p> <pre> >> conversion_d2c_ex2 k=1.002 tau=1.014 xi=0.984 theta=0.551 </pre>

Table 11.2 MATLAB code for the model conversion in Example 11.3.

conversion_d2c_ex3.m	g_discrete_ex3.m
<pre> clear; w=0.0; delta_w=0.05; delt=0.1; %sampling time while(1) % search boundary in which wu exists w=w+delta_w; g=g_discrete_ex3(w,delt); if(imag(g)>0.0) break; end end w1=w-delta_w; w2=w; % w1< wu < w2 while(1) % find wu using the bisection method w=(w1+w2)/2; g1=g_discrete_ex3(w1,delt); g=g_discrete_ex3(w,delt); if(imag(g)*imag(g1)>0.0) w1=w; else w2=w; end if(abs(imag(g))<0.000001) break; end end wu=w; % ultimate frequency wu is found k=abs(g_discrete_ex3(0.0,delt)); g=g_discrete_ex3(wu,delt); tau=sqrt(k^2-abs(g)^2)/abs(g)/wu; theta=(pi+atan(-tau*wu))/wu; % tau: time constant, theta: time delay, k: static gain fprintf('k=%5.3f tau=%5.3f theta=% 5.3f \n',k,tau,theta); </pre>	<pre> function [G]=g_discrete_ex3(w,delt) z=exp(i*w*delt); G=z^(-5)*(0.004443*z^(-1) +0.0044251*z^(-2))/(1-1. 8142*z^(-1)+0.82305*z^(-2)); end </pre>
	<pre> Command Window >> conversion_d2c_ex2 k=1.002 tau=2.320 theta=1.001 </pre>

where $\hat{a}_1 = -1.8142$, $\hat{a}_2 = 0.82305$, $\hat{b}_1 = 0.004443$ and $\hat{b}_2 = 0.0044251$. $\hat{d} = 5$. The sampling time is $\Delta t = 0.1$. From (11.21) and the model parameters, it is clear that the transfer function is

$$G(z) = \frac{\hat{y}(z)}{u(z)} = \frac{\hat{b}_1 z^{-1} + \hat{b}_2 z^{-2}}{1 + \hat{a}_1 z^{-1} + \hat{a}_2 z^{-2}} z^{-\hat{d}} = \frac{0.004443z^{-1} + 0.0044251z^{-2}}{1 - 1.8142z^{-1} + 0.82305z^{-2}} z^{-5} \quad (11.22)$$

Example 11.2

Convert the discrete-time SOPTD model in Example 11.1 to the continuous-time SOPTD model.

Solution The continuous-time SOPTD model by (11.10), (11.12) and (11.17) is

$$G(s) = \frac{\hat{y}(s)}{u(s)} = \frac{1.002 \exp(-0.55s)}{1.014^2 s^2 + 2 \times 1.014 \times 0.984s + 1} \quad (11.23)$$

The MATLAB code for the model conversion from the discrete model to the continuous model is shown in Table 11.1. Note that the continuous-time SOPTD model obtained is very close to the real process in Example 10.3. Theoretically, if the sampling time is smaller, then the continuous-time model obtained is closer to the real process.

Example 11.3

Convert the discrete-time SOPTD model in Example 11.1 to the continuous-time FOPTD model.

Solution The continuous-time FOPTD model by (11.18), (11.19) and (11.20) is

$$G(s) = \frac{\hat{y}(s)}{u(s)} = \frac{1.002\exp(-1.002s)}{2.320s + 1} \quad (11.24)$$

The MATLAB code for the model conversion from the discrete model to the continuous model is shown in Table 11.2.

Problems

- 11.1 Activate the following process using a step input signal with a sampling time of 0.2 and estimate the ARX model with a known time delay. Compare the Nyquist plot of the ARX model and that of the process.

$$\frac{d^3y(t)}{dt^3} + 3\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = u(t - 0.2)$$

- 11.2 Estimate an ARX model with an unknown time delay for the activated process data of Problem 11.1 and compare the Nyquist plot of the ARX model and that of the process.
- 11.3 Estimate the OE models with an unknown time delay for the activated process data of Problem 11.1 and compare the Nyquist plots of the OE models and that of the process.
- 11.4 Convert the discrete-time models of Problems 11.1–11.3 to a continuous-time FOPTD model and tune the PID controller using the IMC tuning rule and show the control performance of the PID controller for the process of Problem 11.1.
- 11.5 Convert the discrete-time models of Problems 11.1–11.3 to a continuous-time SOTPD model and tune the PID controller using the ITAE-2 tuning rule and show the control performance of the PID controller for the process of Problem 11.1.
- 11.6 Activate the virtual process of Process 3 (refer to the Appendix for details) using a step input signal and estimate the ARX model and the unknown time delay. Next, obtain the continuous-time SOPTD model from the discrete-time ARX model and tune the PID controller using the ITAE-2 tuning rule. Finally, show the control performance of the PID controller for Process 3 (refer to the Appendix for details).
- 11.7 Solve Problem 11.6 again with the OE model with an unknown time delay.

Bibliography

- Seborg, D.E., Edgar, T.F. and Mellichamp, D.A. (1989) *Process Dynamics and Control*, John Wiley & Sons, Inc.
- Sung, S.W. and Lee, I. (1996) Limitations and countermeasures of PID controllers. *Industrial & Engineering Chemistry Research*, **35**, 2596.

Part Four

Process Activation

The tuning of a PID controller or the design of an advanced model-based controller goes through the following steps. Step 1, activate the process with a test signal generator. Step 2, estimate the process model using the process identification algorithms. Step 3, tune the PID controller or design the advanced model-based controller. Step 2 and Step 3 are described in Part Three and Part Two respectively. This chapter talks about the test signal generator in Step 1. If the process is activated too aggressively, then the quality of the products from the process may not be acceptable and the safety of the process is not guaranteed. Meanwhile, if activation is not enough, then an accurate process model cannot be obtained because the information included in the activated data is limited and the uncertainties, such as measurement noise and disturbances, become dominant. So, the goals in Step 1 are activating the process in as short a time as possible and activating the dynamic information (frequency components) as much as possible. One of the most efficient methods for process activation is the relay feedback method. If the process is activated with the relay feedback method, then it is straightforward to detect the time-scale (ultimate period) of the process and what frequency components are included in the activated data. Then, it is easy to determine the termination time for the process activation and the design parameters (for example, the maximum frequency, the sampling time, the parameters of the weight) for the process identification methods. So, relay feedback methods and their modifications are introduced in this part.

12

Relay Feedback Methods

In this chapter, the conventional relay feedback method is first introduced. It is the simplest and has been the most widely used in industry for a long time. In particular, it is one of the most important process activation methods for automatic tuning of a PID controller. Next, three important relay feedback methods recently developed to overcome the several problems of the conventional relay feedback methods are introduced. The first method activates the process with a guarantee of a symmetric oscillation under the circumstance of disturbances so that the describing function analysis method provides accurate frequency response data of the process. The second method can guarantee a symmetric oscillation under the circumstance of disturbances and nonlinearity. The third method can provide estimates for the frequency responses for which the phase angle is specified a priori and also manipulate a large range of operation, possibly larger than the magnitude of the relay.

12.1 Conventional Relay Feedback Methods

The conventional relay feedback method was proposed by Åström and Hägglund (1984). It is the simplest among the various versions of relay feedback methods. Two types are available, according to the actual operations: unbiased relay and biased relay.

12.1.1 Unbiased-Relay Feedback Method

Chapter 8 in Part Three briefly explained how to activate a process using the unbiased-relay feedback method, and the MATLAB code for its implementation was introduced. Let us summarize it again. The symbol for an unbiased relay is shown in Figure 12.1 and the block diagram of an unbiased-relay feedback control system is shown in Figure 12.2.

In Figure 12.1, the x -axis and the y -axis represent the input of the relay and the output of the relay respectively. From the symbol in Figure 12.1, it is clear that the relay output is d if the input is greater than zero; otherwise the relay output is $-d$. For example, the relay output will be the square signal in Figure 12.3 if the relay input is the sine signal $a \sin(\omega t)$.

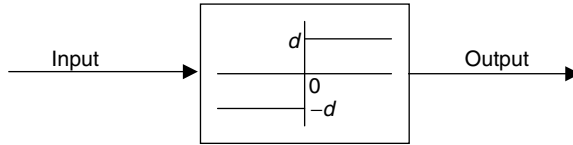


Figure 12.1 Symbol of the unbiased relay.

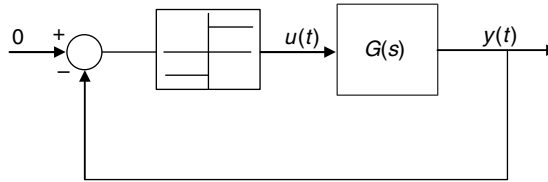


Figure 12.2 Block diagram of the unbiased-relay feedback control system to activate the process output.

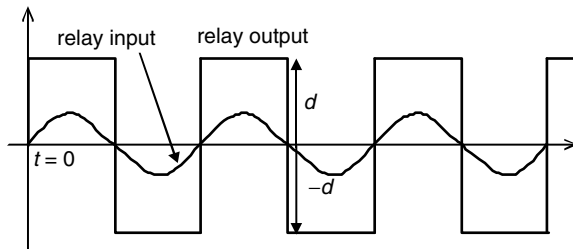


Figure 12.3 Relay output for a sine input.

Now, consider the relay feedback control system in Figure 12.2 to activate the process. It should be noted that the relay input is $0 - y(t)$, where $y(t)$ is the process output. So, the upper value d of the relay is applied to the process when the process output is less than the reference value of zero, and vice versa. That is, $u(t) = d$ if $y(t) \leq 0$ and $u(t) = -d$ if $y(t) > 0$.

Figure 12.4 shows the activated process input and the process output by the unbiased-relay feedback method, where $y_{\text{ref}}(t) = 0$ because it is the unbiased-relay feedback method.

The detailed procedure for the unbiased-relay feedback method is as follows. First, the upper (on) value of the relay output is applied to drag the process output out of the initial value, as shown in Figure 12.4. Second, the lower (off) value of the relay is applied when the process output deviates from the initial state. Third, the upper value of the relay is applied when the process output is less than the reference value, and vice versa. That is, $u(t) = d$ if $y(t) \leq 0$ and $u(t) = -d$ if $y(t) > 0$. Then, the process input and output usually reach a cyclic steady state (which means that the period and the peak value of the process output do not change) after three or four cycles.

The remarkable properties of the unbiased-relay feedback method are summarized as follows. First, the time-length of the process activation is automatically determined by the three or four on–offs of the relay. Second, it has no tuning parameters except the magnitude of the relay. Third, the main frequency component included in the process input and output of

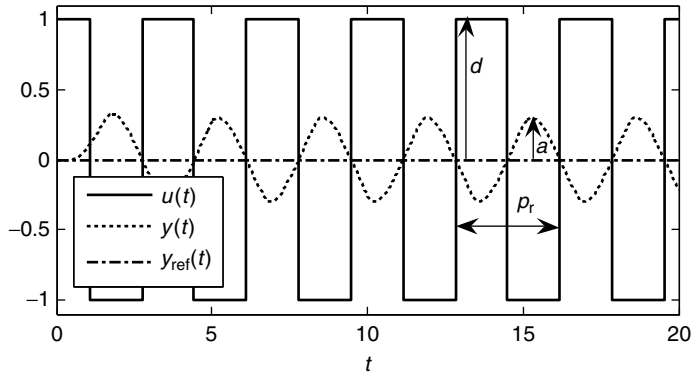


Figure 12.4 Activated process output by the unbiased-relay feedback method.

the cyclic-steady-state part is the fundamental frequency component. Fourth, the frequency of the cyclic steady state is very close to the ultimate frequency of the process. The first property and the second property make the activation method the simplest and easiest in implementation. The third property says that only the fundamental frequency response model from the cyclic-steady-state part can be estimated.

Example 12.1

Choose all the processes to which the unbiased-relay feedback method can be applied.

- (P1) $G(s) = \exp(-0.5s)/(s+1)^2$
- (P2) $G(s) = \exp(-0.2s)/(s+1)$
- (P3) $G(s) = 1/(s+1)^3$
- (P4) $G(s) = 1/(s+1)$
- (P5) $G(s) = (-0.2s+1)/(s+1)^5$
- (P6) $G(s) = 1/(s+1)^2$

Solution Processes P1, P2, P3 and P5 have the ultimate frequencies, but P4 and P6 have no ultimate frequencies. So, the unbiased relay cannot be applied to processes P4 and P6. The unbiased-relay feedback system will produce a cycle in which the period converges to zero, resulting in no cyclic-steady-state cycling for processes P4 and P6.

Example 12.2

Activate the process $G(s) = \exp(-0.2s)/(s+1)^2$ using the unbiased-relay feedback method.

Solution The activated process input and output and the MATLAB code to activate the process are shown Figure 12.5 and Table 12.1 respectively.

Example 12.3

Activate the process $G(s) = \exp(-0.2s)/(s+1)^2$ using the unbiased-relay feedback method when the process output is contaminated with uniformly distributed random noise between -0.1 and 0.1 .

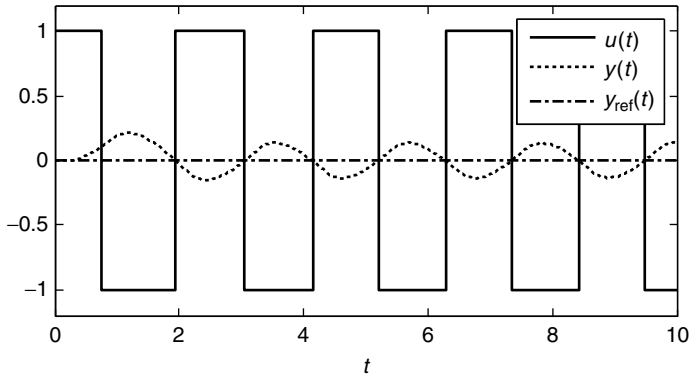


Figure 12.5 Activated process output by the unbiased-relay feedback method.

Table 12.1 MATLAB code to simulate Figure 12.5.

<pre> unbiased_relay_ex2.m clear; delt=0.01; tf=10; n=round(tf/delt); x=zeros(2,1); u_data=zeros(1,500); t_on=0.0; t_off=0.0; P_on=0; P_off=0; ymin=0.0; ymax=0.0; y=0.0; yref=0.0; index=0; y_delta=0.1; d=1.0; % initial phase:index=0, relay phase:index=1 for i=1:n t=i*delt; yy(i)=y; yyref(i)=yref; tt(i)=t; if(index==1) if(yy(i)>yref & yy(i-1)<=yref) P_on=t-t_on; t_off=t; ymax_f=ymax; ymax=0.0; end if(yy(i)<=yref & yy(i-1)>yref) P_off=t-t_off; t_on=t; ymin_f=ymin; ymin=0.0; end end if(yy(i)>yref) u=-d; if(yy(i)>ymax) ymax=yy end if(yy(i)<=yref) u=d; if(yy(i)<ymin) ymin=yy(i); end end end </pre>	<pre> g_unbiased_relay_ex2.m function [next_x,y]=g_unbia- sed_relay_ex2(x,delt,u) subdelt=delt; n=round(delt/ subdelt); A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.2; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return end </pre> <p>command window</p> <pre> >> unbiased_relay_ex2 Period = 2.1400 Peak Value = 0.1344 Relay Magnitude = 1.0000 </pre>
---	--

Table 12.1 (Continued).

<pre>if(index==0) u=d; if(yy(i)>y_delta) index=1; end end for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u; uu(i)=u; [x,y]=g_unbiased_relay_ex2(x, delt,u_data); end P=P_on+P_off; a=(abs(ymax_f)+abs (ymin_f))/2; fprintf('Period=%7.4f Peak Value=% 7.4f Relay Magnitude=%7.4f\n',P,a, d); figure(1); plot(tt,uu,tt,yy,tt,yyr- ef);]]></pre>	
--	--

Solution The activated process input and output and the MATLAB code to activate the process are shown Figure 12.6 and Table 12.2 respectively. In this example, the readers should pay attention to the technique to manipulate the measurement noise. If the MATLAB code of Table 12.1 is directly applied to the case of the measurement noise, then the relay output will show severe fluctuations around the zero-crossing point. So, the hysteresis is used to prevent the phenomenon, as shown in Table 12.2. The frequency of the oscillation moves to a lower frequency region compared with the ultimate frequency if the hysteresis is used. Note that the period in Table 12.2 is longer than that in Table 12.1 because of the hysteresis.

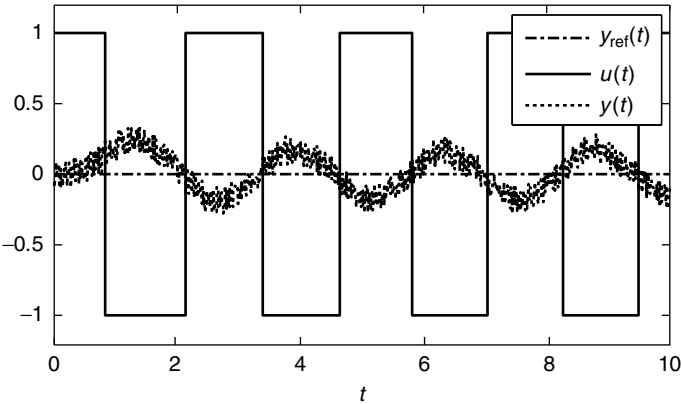


Figure 12.6 Activated process output by the unbiased-relay feedback method for the case of measurement noise.

Table 12.2 MATLAB code to simulate Figure 12.6.

unbiased_relay_ex3.m	g_unbiased_relay_ex3.m
<pre> clear; delt=0.01; tf=10; n=round(tf/delt); u_data=zeros(1,500); x=zeros(2,1); t_on=0.0; t_off=0.0; P_on=0; P_off=0; y=0.0; yref=0.0; np=0; index=0; y_delta=0.2; d=1.0; % initial phase:index=0, relay phase:index=1 hys=0.1; index_up=1; index_down=0; ymin=0.0; ymax=0.0; rand('seed',0); noise=(rand (1,n)-0.5)*0.2; for i=1:n t=i*delt; yy(i)=y+noise (i); yyref(i)=yref; tt(i) =t; if(index==1) if(index_down==1 & index_up==0 & yy(i)<=(yref- hys) & yy(i-1)>(yref-hys)) index_up=1; index_down=0; ymin_f=ymin; ymin=0.0; t_on=t; P_off=t_on- t_off; end if(index_up==1 & index_down==0 & yy(i)>(yref+hys) & yy(i-1) <=(yref+hys)) index_up=0; index_down=1; ymax_f=ymax; ymax=0.0; t_off=t; P_on=t_off-t_on; np=np+1 end end if(index_down==1) u=-d; if(yy(i)>ymax) ymax=yy(i); end end if(index_up==1) </pre>	<pre> function [next_x,y]=g_unbiased_relay_ex3 (x,delt,u); subdelt=delt; n=round(delt/ subdelt); A=[0 -1;1 -2]; B=[1;0]; C=[0 1]; delay=0.2; delay_k=round(delay/delt +0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return end </pre>
	<pre> command window >> unbiased_relay_ex3 Period = 2.4600 Peak Value = 0.2653 Relay Magnitude = 1.0000 </pre>

Table 12.2 (Continued)

<pre> u=d; if (yy(i)<ymin) ymin=yy(i); end end if (index==0) u=d; if (yy(i)>y_delta) index=1; if (yref<y_delta) u=-d; index_up=0; index_down=1; end end end for j=1:499 u_data(j)=u_data(j+1); end u_data(500)=u; uu(i)=u; P=P_on+P_off; [x,y]= g_unbiased_ relay_ex3 (x,delt,u_data); end P=P_on+P_off; a=(abs (ymax_f)+abs(ymin_f))/2; fprintf(' Period = %7.4f Peak Value = %7.4f Relay Magnitude = %7.4f \n',P,a,d); figure(1); plot(tt,yyref,tt, uu,tt,yy);</pre>	
--	--

12.1.2 Biased-Relay Feedback Method

Chapter 8 in Part Three briefly explained how to activate the process using the biased-relay feedback method, and the MATLAB code for its implementation was introduced. Let us summarize it again. The block diagram of the biased-relay feedback control system is shown in Figure 12.7 (Shen *et al.*, 1996a).

Now, consider the relay feedback control system in Figure 12.7 to activate the process. It should be noted that the relay input is $y_{ref}(t) - y(t)$, where $y(t)$ is the process output and $y_{ref}(t)$ is the reference value for the relay on–off. So, the upper value d of the relay is applied to the process when the process output is less than the reference value $y_{ref}(t)$, and vice versa. That is, $u(t) = d$ if $y(t) \leq y_{ref}(t)$ and $u(t) = -d$ if $y(t) > y_{ref}(t)$.

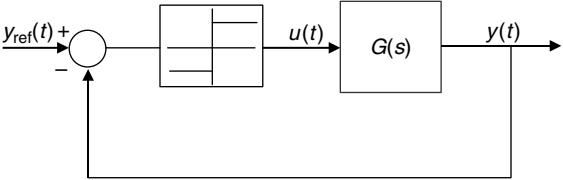


Figure 12.7 Block diagram of the biased-relay feedback control system to activate the process output.

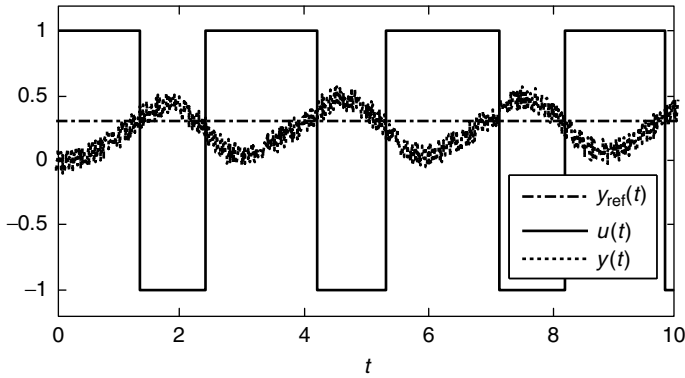


Figure 12.8 Activated process output by the unbiased-relay feedback method with $y_{\text{ref}}(t) = 0.3$.

Figure 12.8 shows the activated process input and the process output by the biased-relay feedback method with $y_{\text{ref}}(t) = 0.3$.

The remarkable properties of the biased-relay feedback method are summarized as follows. First, the time-length of the process activation is automatically determined by the three or four on-offs of the relay. Second, it has no tuning parameters except the magnitude of the relay and the reference value. Third, the frequency components included in the cyclic-steady-state part are the two frequency components corresponding to zero and the fundamental frequency. Meanwhile, the initial (unsteady-state) part of the activated process input and the process output includes various frequency components. Fourth, the frequency of the cyclic steady state is different from the ultimate frequency of the process. The fundamental frequency of the biased-relay feedback control system is lower than that of the unbiased-relay feedback control system. The first property and the second property make the activation method the simplest and easiest in implementation. The third property implies that only the two frequency response data of the zero and fundamental frequency from the cyclic-steady-state part can be identified, while many other frequency response data from the initial (unsteady-state) part of the process input and output can theoretically be obtained. The fourth property means that the ultimate frequency response data from the cyclic-steady-state part cannot be estimated.

Example 12.4

Simulate Figure 12.6 again with $y_{\text{ref}}(t) = 0.3$.

Solution The simulation results are shown in Figure 12.8. It is straightforward to obtain the simulation results in Figure 12.8 by replacing $y_{\text{ref}} = 0.0$ in Table 12.2 by $y_{\text{ref}} = 0.3$.

12.2 Relay Feedback Method to Reject Static Disturbances

Consider the process input and the process output in Figure 12.8. Clearly, the process input $u(t)$ is asymmetric (that is, the two half periods are totally different) because $y_{\text{ref}}(t)$ is not zero. This phenomenon results in two problems. First, the fundamental frequency is far from the ultimate frequency. Second, the harmonics terms become too large to be neglected. Then, the describing function analysis method cannot provide acceptable accuracy in estimating the frequency

response model for the ultimate frequency. Fourier analysis can estimate the frequency response exactly for the fundamental frequency, but the estimate is not for the ultimate frequency because the fundamental frequency is far from the ultimate frequency.

In this section, a relay feedback method is introduced to prevent the phenomenon. Consider the relay feedback control system in Figure 12.9 (Shen *et al.*, 1996b; Park *et al.*, 1997). It removes the effects of disturbances or input reference value $y_{\text{ref}}(t)$ by adjusting the output reference value $u_{\text{ref}}(t)$ for the relay on–off as much as the disturbance. Here, $u_r(t)$ and $u(t)$ are the relay output and the process input respectively.

Park *et al.* (1997) and Shen *et al.* (1996b) proposed the following update rule for $u_{\text{ref}}(t)$:

$$u_{\text{ref},k} = u_{\text{ref},k-1} - \alpha \left[\frac{(a_{\text{max},k-1} + a_{\text{min},k-1})d}{|a_{\text{max},k-1}| + |a_{\text{min},k-1}|} \right] \quad (12.1)$$

where $a_{\text{max},k-1}$ and $a_{\text{min},k-1}$ are the $(k-1)$ -th peak value and the $(k-1)$ -th valley value of the process output. $u_{\text{ref},k}$ is the k th reference value. α is the tuning parameter to compromise between the convergence rate and the robustness.

Figure 12.10 shows the responses of the process $G(s) = \exp(-0.2s)/(s+1)^2$ controlled by the relay feedback method for static disturbance rejection. A static disturbance of 0.3 is added to the relay output from the beginning of the relay feedback test. The update of $u_{\text{ref}}(t)$ starts from the second cycling. The tuning parameter is chosen as $\alpha = 0.5$. As expected, $u(t)$ and $y(t)$ become symmetric in the cyclic steady state, as shown in Figure 12.10, because the update rule of (12.1) adjusts $u_{\text{ref}}(t)$ as much as the static disturbance. Also, the update rule provides an acceptable convergence rate. The MATLAB code to simulate Figure 12.10 is shown in Table 12.3.

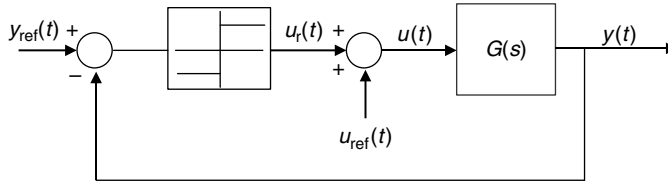


Figure 12.9 Block diagram of the relay feedback control system to remove static disturbances.

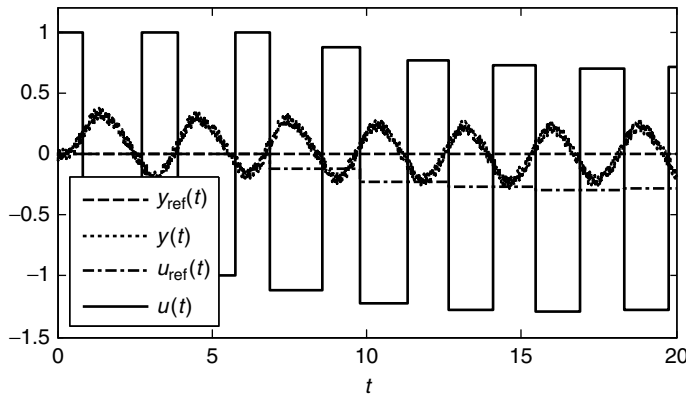


Figure 12.10 Response of the process for the relay feedback method to remove static disturbances.

Table 12.3 MATLAB code to simulate Figure 12.10.

```

                                relay_D1.m
clear; delt=0.01; tf=20; n=round(tf/delt);
u_data=zeros(1,500); x=zeros(2,1);
t_on=0.0; t_off=0.0; P_on=0; P_off=0;
y=0.0; yref=0.0; np=0; index=0; y_delta=0.2; d=1.0; dis=0.3;
% initial phase:index=0, relay phase:index=1
hys=0.05; index_up=1; index_down=0; ymin=0.0; ymax=0.0; uref=0.0;
rand('seed',0); noise=(rand(1,n)-0.5)*0.05;
for i=1:n
    t=i*delt; yy(i)=y+noise(i); yyref(i)=yref; tt(i)=t;
    if(index==1)
        if(index_down==1 & index_up==0 & yy(i)<=(yref-hys) & yy(i-1)>
(yref-hys))
            index_up=1; index_down=0; ymin_f=ymin; ymin=0.0;
            t_on=t; P_off=t_on-t_off;
        end
        if(index_up==1 & index_down==0 & yy(i)>(yref+hys) & yy(i-1)<=(yref
+hys))
            index_up=0; index_down=1; ymax_f=ymax; ymax=0.0;
            t_off=t; P_on=t_off-t_on; np=np+1;
            if(np>=2)
                uref=uref-0.5*(ymax_f+ymin_f)*d/(abs(ymax_f)+abs(ymin_f));
            end
        end
    end
    if(index_down==1)
        ur=-d; if(yy(i)>ymax) ymax=yy(i); end
    end
    if(index_up==1)
        ur=d; if(yy(i)<ymin) ymin=yy(i); end
    end
    if(index==0)
        ur=d;
        if(yy(i)>y_delta)
            index=1;
            if(yref<y_delta) ur=-d; index_up=0; index_down=1; end
        end
    end
    for j=1:499 u_data(j)=u_data(j+1); end
    u_data(500)=ur+uref+dis; uu(i)=ur+uref; uuref(i)=uref;
P=P_on+P_off;
    [x,y]=g_relay_D1(x,delt,u_data);
end
P=P_on+P_off; a=(abs(ymax_f)+abs(ymin_f))/2;
fprintf('Period=%7.4f Peak Value=%7.4f Relay Magnitude=%7.4f\n',P,a,
d);
figure(1); plot(tt,yyref,tt,yy,tt,uuref,tt,uu);

```

Table 12.3 (Continued)

<pre>g_relay_D1.m function [next_x,y]=g_relay_D1(x,delt,u); subdelt=delt; n=round(delt/subdelt); A=[0 -1;1 -2]; B=[1;0]; C=[0 1]; delay=0.2; delay_k=round(delay/delt+0.00001); for i=1:n dx=A*x+B*u(500-delay_k); x=x+dx*subdelt; end next_x=x; yo=C*x; y=yo; return end</pre>
<pre>command window >> relay_D1 Period = 2.6300 Peak Value = 0.2225 Relay Magnitude = 1.0000</pre>

The relay feedback method to reject the disturbance can be applied to the biased-relay feedback method. As shown in Figure 12.8, the biased relay with $y_{\text{ref}}(t) = 0.3$ cannot provide a symmetric oscillation, resulting in larger harmonics. This problem can be easily solved by defining a new variable of $y_{\text{new}}(t) = y(t) - y_{\text{ref}}(t)$ and applying the relay feedback method for the disturbance rejection to the new variable of $y_{\text{new}}(t)$ instead of the process output $y(t)$. As shown in Figure 12.11, it successfully obtains a symmetric oscillation at $y_{\text{ref}}(t) = 0.3$. The MATLAB code to simulate Figure 12.11 is shown in Table 12.4.

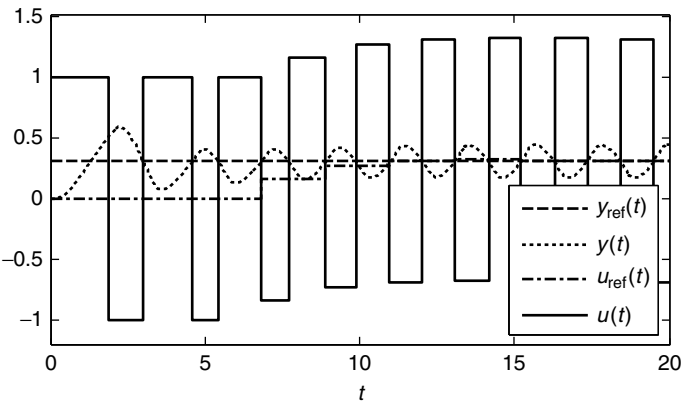


Figure 12.11 Response of the process for the relay feedback method to remove static disturbances with $y_{\text{ref}}(t) = 0.3$.

Table 12.4 MATLAB code to simulate Figure 12.11.

```

                                relay_D2.m
clear; delt=0.01; tf=20; n=round(tf/delt);
u_data=zeros(1,500); x=zeros(2,1);
t_on=0.0; t_off=0.0; P_on=0; P_off=0;
y=0.0; yref_new=0.0; yref=0.3; np=0; index=0; y_delta=0.2; d=1.0;
% initial phase:index=0, relay phase:index=1
index_up=1; index_down=0; ymin=0.0; ymax=0.0; uref=0.0;
for i=1:n
    t=i*delt; yy_new(i)=y-yref; yy(i)=y; yyref(i)=yref; tt(i)=t;
    if(index==1)
        if(index_down==1 & index_up==0 & yy_new(i)<=yref_new & yy_new(i-1)
>yref_new)
            index_up=1; index_down=0; ymin_f=ymin; ymin=0.0;
            t_on=t; P_off=t_on-t_off;
        end
        if(index_up==1 & index_down==0 & yy_new(i)>yref_new & yy_new(i-1)
<=yref_new)
            index_up=0; index_down=1; ymax_f=ymax; ymax=0.0;
            t_off=t; P_on=t_off-t_on; np=np+1;
            if(np>=2) uref=uref-0.4*(ymax_f+ymin_f)*d/(abs(ymax_f)+abs
(ymin_f)); end
        end
    end
    if(index_down==1)
        ur=-d; if(yy_new(i)>ymax) ymax=yy_new(i); end
    end
    if(index_up==1)
        ur=d; if(yy_new(i)<ymin) ymin=yy_new(i); end
    end
    if(index==0)
        ur=d;
        if(yy_new(i)>y_delta)
            index=1;
            if(yref_new<y_delta) ur=-d; index_up=0; index_down=1; end
        end
    end
    for j=1:499 u_data(j)=u_data(j+1); end
    u_data(500)=ur+uref; uu(i)=ur+uref; uuref(i)=uref; P=P_on+P_off;
    [x,y]=g_relay_D2(x,delt,u_data);
end
P=P_on+P_off; a=(abs(ymax_f)+abs(ymin_f))/2;
fprintf('Period = %7.4f Peak Value = %7.4f Relay Magnitude = %7.4f \n',P,a,
d);
figure(1); plot(tt,yyref,tt,yy,tt,uuref,tt,uu);figure(1); plot(tt,yyr-
ef,tt,yy,tt,uuref,tt,uu);

```

```

                                g_relay_D2.m
function [next_x,y]=g_relay_D2(x,delt,u);

```

Table 12.4 (Continued)

```

subdelt=delt; n=round(delt/subdelt);
A=[0 -1;1 -2]; B=[1;0];
C=[0 1]; delay=0.2;
delay_k=round(delay/delt+0.00001);
for i=1:n
    dx=A*x+B*u(500-delay_k);
    x=x+dx*subdelt;
end
next_x=x; yo=C*x; y=yo;
return

```

command window

```

>> relay_D2
Period = 2.1500 Peak Value = 0.1342 Relay Magnitude = 1.0000

```

12.3 Relay Feedback Method under Nonlinearity and Static Disturbances

The relay feedback method introduced in this section is used to manipulate output nonlinearities and static disturbances. Let us call it Relay_ND. It guarantees the symmetry of the relay output by setting the time length of the relay off to the half period of the previous cycle. And it rejects the effects of static disturbances and output nonlinearity by changing the input reference value of the relay. Relay_ND can be successfully applied to identify a Wiener-type nonlinear process with a static disturbance.

It activates the process using the following algorithm (Sung and Lee, 2006):

$$u(t) = -d \quad \text{for } t_{\text{off},k} \leq t < t_{\text{off},k} + \frac{P_{k-1}}{2} \quad (12.2)$$

$$t_{\text{on},k} = t_{\text{off},k} + \frac{P_{k-1}}{2}, \quad y_{\text{ref},k} = y(t_{\text{on},k}) \quad \text{for } t = t_{\text{off},k} + \frac{P_{k-1}}{2} \quad (12.3)$$

$$u(t) = d \quad \text{for } t_{\text{off},k} + \frac{P_{k-1}}{2} \leq t < t_{\text{off},k+1} \quad (12.4)$$

$$t_{\text{off},k+1} = t, \quad P_k = t_{\text{off},k+1} - t_{\text{off},k} \quad \text{for } t \geq t_{\text{off},k} + \frac{1.5P_{k-1}}{2} \text{ and } y(t) \geq y_{\text{ref},k} \quad (12.5)$$

where $u(t)$ and $y(t)$ are the relay output and the process output respectively. $y_{\text{ref},k}$, P_k , $t_{\text{off},k}$ and $t_{\text{on},k}$ are the reference value, period, time for relay off and time for relay on of the k th cycle. The algorithm can be represented graphically as shown in Figure 12.12.

The remarkable difference between the conventional relay and Relay_ND is that Relay_ND uses the half period of the previous cycle to determine the relay off, which produces a symmetric cycling, while the conventional relay uses the point at which the process output crosses the reference value, which means that it has no equipment to enforce a symmetric cycling. Equation (12.4) enforces the symmetry of the relay output in the cyclic steady state. $y_{\text{ref},k} = y(t_{\text{on},k})$ in (12.3) makes the two crossing points in one cycle between the process output and the reference value converge to the same value, which rejects the effects of static

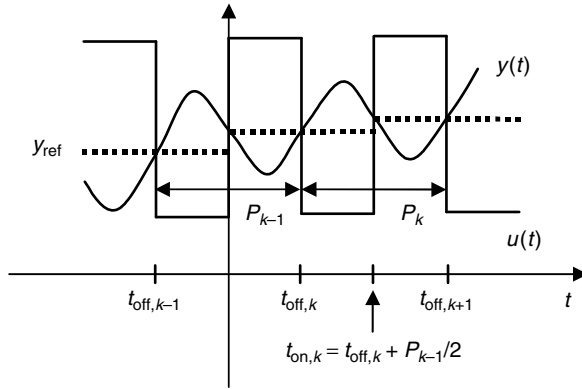


Figure 12.12 Relay feedback method under nonlinearity and static disturbance conditions.

disturbances. For the case of severe measurement noise, it is recommended that $y_{\text{ref},k} = a_1 + a_2 P/2$, where a_1 and a_2 are the estimates of the least-squares method of which the object function is

$$\min_{a_1, a_2} \sum_{i=1}^N (y(t_i) - a_1 - a_2(t_i - t_{\text{off}}))^2$$

subject to

$$P/2 - \alpha P \leq t_i - t_{\text{off}} \leq P/2$$

P and t_{off} are the previous period and the time corresponding to the recent relay off. The effects of the measurement noise decrease as α increases, but α should be small enough for y to be approximately linear with respect to t within the time span $P/2 - \alpha P \leq t - t_{\text{off}} \leq P/2$.

Example 12.5

Simulate Relay_ND for the case of no noise ($\alpha = 0$) and a static input disturbance $d = -0.5$ with the process $y(s)/u(s) = \exp(-0.5s)/(s + 1)^2$.

Solution The MATLAB code to simulate Relay_ND and the results are shown in Table 12.5 and Figure 12.13 respectively. The estimate for the ultimate period is close to the true value of 3.27 even under the circumstance of the static disturbance.

Example 12.6

Simulate Relay_ND for the case of measurement noise ($\alpha = 0.1$) and a static disturbance $d = -0.5$ with the process $y(s)/u(s) = \exp(-0.5s)/(s + 1)^2$. The measurement noise is uniformly distributed random noise between -0.05 and 0.05 .

Solution The MATLAB code to simulate Relay_ND and the result are shown in Table 12.6 and Figure 12.14 respectively. It shows acceptable robustness to the measurement noise.

Table 12.5 MATLAB code to simulate Example 12.5.

```

                                relay_ND1.m
clear;
t=0; tf=35.0; delt=0.01; n=round(tf/delt);
y=0; x=[0;0]; delay=0.5; n_delay=round(delay/delt);
u=zeros(1,500); u_relay=0; dis=-0.5; d=1.0;
yref=0.0; yref_on=0;
tref=0.0; toff_ref=0.0; ton_ref=0.0;
m=0; index=0; y_delta=0.1;
for k=1:n
    u_relay_b=u_relay; % one sampling before : u_relayb
    tp=t-tref; toff=t-toff_ref; ton=t-ton_ref;
    if(index==0)
        u_relay=d; toff_ref=t; p=t; if(y>y_delta) index=1; end
    else
        if(m<2) % conventional relay
            if((y>yref) & (ton>p/4)) u_relay=-d; end
            if((y<yref) & (toff>p/4)) u_relay=d; end
        else % relay_ND
            if(tp<p/2) u_relay=-d; end
            if(tp>=p/2) u_relay=d; end
            if((tp>=1.5*p/2) & (y>yref)) u_relay=-d; end
            yref=yref_on;
        end
        if((u_relay==d) & (u_relay_b==d)) % when relay on
            ton_ref=t; yref_on=y; sumy=0.0;
        end
        if((u_relay==-d) & (u_relay_b==d)) % when relay off
            m=m+1; p=tp; tref=t; toff_ref=t;
        end
    end
    for i=1:499 u(i)=u(i+1); end
    u(500)=u_relay+dis;
    um(k)=u_relay; tm(k)=t; ym(k)=y; yr(k)=yref;
    [x,y]=g_relay_ND1(x,u(500-n_delay),delt);
    t=t+delt;
    end
    fprintf('Pu=%5.2f y_ref=%5.2f \n',p,yref);
    figure(1); plot(tm,ym,tm,um,tm,yr);

```

```

                                g_relay_ND1.m
function [x,y]=g_relay_ND1(x,u_delay,delt)
    A=[0 -1 ; 1 -2 ]; B=[1 ; 0 ]; C=[0 1];
    x=x+(A*x+B*u_delay)*delt;
    y=C*x;
return

```

```

                                command window

>> relay_ND1
Pu= 3.32 y_ref=-0.51

```

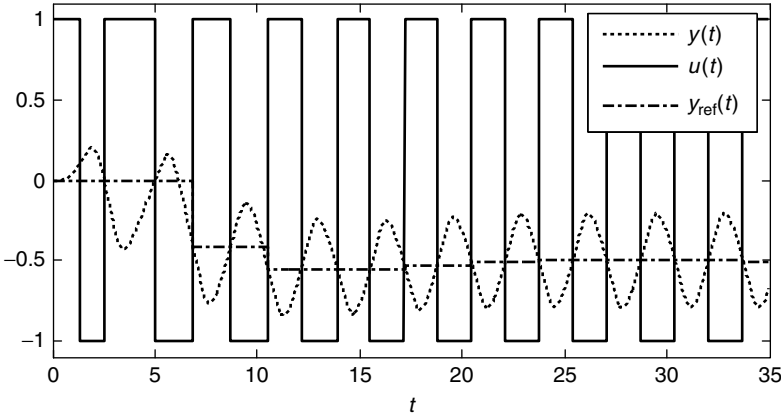


Figure 12.13 Simulation results in Example 12.5.

Example 12.7

Estimate the ultimate frequency data of the linear dynamic subsystem (12.7) and the output nonlinear static function (12.8) using Relay_ND for the following Wiener process:

$$w(t) = u(t) + 0.15 \quad (12.6)$$

$$\frac{z(s)}{w(s)} = G(s) = \frac{\exp(-0.5s)}{(s+1)^2} \quad (12.7)$$

$$y(t) = 1 - (1 + z(t)/3.0)\exp(-3.0z(t)) \quad (12.8)$$

The process output is contaminated by uniformly distributed random noise between -0.05 and 0.05 .

Solution The activated process output by Relay_ND is shown in Figure 12.15 and the MATLAB code is shown in Table 12.7.

The estimate for the ultimate period is close to the true value of 3.27 even under the circumstance of the static disturbance and output nonlinearity. Now, let us estimate the output nonlinear static function. The relay output is known to be approximately $u(t) = (4d/\pi)\sin(\omega t)$. Here, ω is the relay frequency. Then, $z(t)$ in the cyclic steady state is approximately $z(t) \approx 0.15G(0) - (4d/\pi)|G(i\omega)|\sin(\omega t)$. Let us parameterize the model for the output nonlinear function in the form $z(t) = \hat{g}_1 y + \hat{g}_2 y^2 + \hat{g}_3 y^3 + \cdots + \hat{g}_n y^n$. Then, the normalized model parameters of \hat{f}_k , $k = 1, 2, \dots, n$, can be obtained by solving the following optimization problem using the least-squares method:

$$\begin{aligned} & \min_{\hat{g}} \sum_{k=1}^N [-0.15G(0) + (4d/\pi)|G(i\omega)|\sin(\omega t_k) + \hat{g}_1 y_k + \hat{g}_2 y_k^2 + \hat{g}_3 y_k^3 + \cdots + \hat{g}_n y_k^n]^2 \\ &= \min_{\hat{g}} \sum_{k=1}^N \left[\sin(\omega t_k) - \frac{0.15G(0)}{(4d/\pi)|G(i\omega)|} + \frac{\hat{g}_1 y_k + \hat{g}_2 y_k^2 + \hat{g}_3 y_k^3 + \cdots + \hat{g}_n y_k^n}{(4d/\pi)|G(i\omega)|} \right]^2 \\ &= \min_{\hat{f}} \sum_{k=1}^N [\sin(\omega t_k) + \hat{f}_0 + \hat{f}_1 y_k + \hat{f}_2 y_k^2 + \hat{f}_3 y_k^3 + \cdots + \hat{f}_n y_k^n]^2 \end{aligned} \quad (12.9)$$

Table 12.6 MATLAB code to simulate Example 12.6.

```

                                relay_ND2.m
clear;
t=0; tf=35.0; delt=0.01; n=round(tf/delt);
y=0; x=[0;0]; delay=0.5; n_delay=round(delay/delt);
u=zeros(1,500); u_relay=0; dis=-0.5; d=1.0;
yref=0.0; yref_on=0;
tref=0.0; toff_ref=0.0; ton_ref=0.0;
m=0; index=0; y_delta=0.2;
rand('seed',0); noise=(rand(1,n)-0.5)*0.1;
s1=0; s2=0; s3=0; s4=0; s5=0;
for k=1:n
    u_relay_b=u_relay; % one sampling before : u_relayb
    tp=t-tref; toff=t-toff_ref; ton=t-ton_ref;
    if(index==0)
        u_relay=d; toff_ref=t; p=t; if(y>y_delta) index=1; end
    else
        if(m<2) % conventional relay
            if((y>yref) & (ton>p/4)) u_relay=-d; end
            if((y<yref) & (toff>p/4)) u_relay=d; end
        else % relay_ND
            if(tp<p/2) u_relay=-d; end
            if(tp>=p/2) u_relay=d; end
            if((tp>=1.5*p/2) & (y>yref)) u_relay=-d; end
            if((tp>=(p/2-p*0.1)) & (tp<(p/2)))
                s1=s1+1; s2=s2+tp; s3=s3+tp^2;
                s4=s4+y; s5=s5+y*tp;
            end
            yref=yref_on;
        end
        if((u_relay==d) & (u_relay_b==d)) % when relay off
            m=m+1; p=tp; tref=t; toff_ref=t;
        end
        if((u_relay==d) & (u_relay_b==d)) % when relay on
            if(m>=2)
                theta=inv([s1 s2; s2 s3])*[s4;s5];
                yref_on=theta(1)+theta(2)*p/2;
            end
            s1=0; s2=0; s3=0; s4=0; s5=0; ton_ref=t;
        end
        for i=1:499 u(i)=u(i+1); end
        u(500)=u_relay+dis;
        um(k)=u_relay; tm(k)=t; ym(k)=y; yr(k)=yref;
        [x,yo]=g_relay_ND2(x,u(500-n_delay),delt);
        y=yo+noise(k); t=t+delt;
    end
    fprintf('Pu=%5.2f y_ref=%5.2f \n',p,yref);
    figure(1); plot(tm,ym,tm,um,tm,yr);

```

Table 12.6 (Continued)

g_relay_ND2.m
function [x,y]=g_relay_ND2(x,u_delay,delt)
A=[0 -1 ; 1 -2]; B=[1 ; 0]; C=[0 1];
x=x+(A*x+B*u_delay)*delt;
y=C*x;
return

command window
>> relay_ND2
Pu= 3.28 y_ref=-0.49

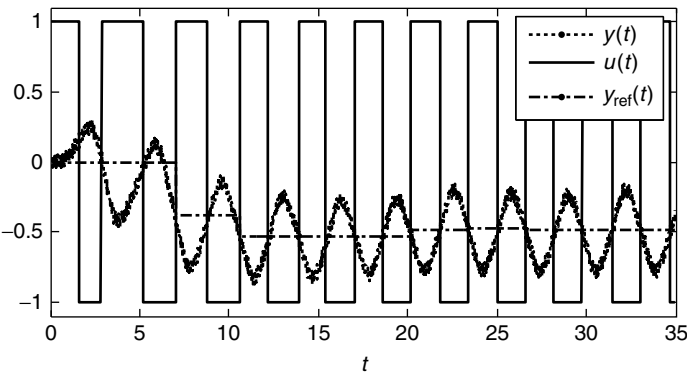


Figure 12.14 Simulation results in Example 12.6.

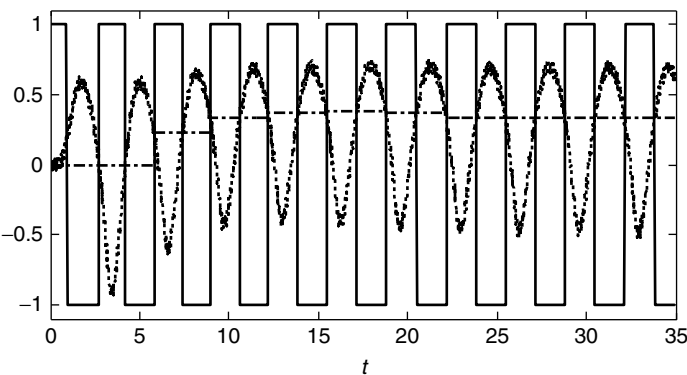


Figure 12.15 Activation by Relay_ND for a Wiener-type nonlinear process.

Table 12.7 MATLAB code to simulate the case of a static disturbance plus measurement noise in Example 12.7.

```

                                relay_ND3.m

clear;
t=0; tf=35.0; deltt=0.01; n=round(tf/deltt);
y=0; x=[0;0]; delay=0.5; n_delay=round(delay/deltt);
u=zeros(1,500); u_relay=0; dis=0.15; d=1.0;
yref=0.0; yref_on=0;
tref=0.0; toff_ref=0.0; ton_ref=0.0;
m=0; index=0; y_delta=0.2;
rand('seed',0); noise=(rand(1,n)-0.5)*0.1;
s1=0; s2=0; s3=0; s4=0; s5=0; j=0;
for k=1:n
    u_relay_b=u_relay; % one sampling before : u_relayb
    tp=t-tref; toff=t-toff_ref; ton=t-ton_ref;
    if(index==0)
        u_relay=d; toff_ref=t; p=t; if(y>y_delta) index=1; end
    else
        if(m<2) % conventional relay
            if((y>yref) & (ton>p/4)) u_relay=-d; end
            if((y<yref) & (toff>p/4)) u_relay=d; end
        else % relay_ND
            if(m==10) j=j+1; data_y_u(j,1:3)=[t y u_relay]; end
            if(tp<p/2) u_relay=-d; end
            if(tp>=p/2) u_relay=d; end
            if((tp>=1.5*p/2) & (y>yref)) u_relay=-d; end
            if((tp>=(p/2-p*0.1)) & (tp<(p/2)))
                s1=s1+1; s2=s2+tp; s3=s3+tp^2;
                s4=s4+y; s5=s5+y*tp;
            end
            end
            yref=yref_on;
        end
        if((u_relay==d)&(u_relay_b==d)) % when relay off
            m=m+1; p=tp; tref=t; toff_ref=t;
        end
        if((u_relay==d)&(u_relay_b==d)) % when relay on
            if(m>=2)
                theta=inv([s1 s2 ; s2 s3])*[s4;s5];
                yref_on=theta(1)+theta(2)*p/2;
            end
            s1=0; s2=0; s3=0; s4=0; s5=0; ton_ref=t;
        end
    end
    for i=1:499 u(i)=u(i+1); end
    u(500)=u_relay+dis;
    um(k)=u_relay; tm(k)=t; ym(k)=y; yr(k)=yref;
    [x,yo]=g_relay_ND3(x,u(500-n_delay),deltt);
    y=yo+noise(k); t=t+deltt;
end

```

Table 12.7 (Continued)

```
fprintf('Pu=%5.2f y_ref=%5.2f \n',p,yref);
figure(1); plot(tm,ym,tm,um,tm,yr);
```

```
g_relay_ND3.m
function [x,y]=g_relay_ND3(x,u_delay,delt)
    A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1];
    x=x+(A*x+B*u_delay)*delt;
    z=C*x; y=1-(1+z/3.0)*exp(-3.0*z);
return
```

command window

```
>> relay_ND3
Pu= 3.33 y_ref= 0.34
```

where the normalized model parameters $\hat{f}_0, \hat{f}_1, \dots, \hat{f}_n$ correspond to $-0.15G(0)/(4d|G(i\omega)|/\pi)$, $\hat{g}_1(4d|G(i\omega)|/\pi)$, \dots , $\hat{g}_n(4d|G(i\omega)|/\pi)$ respectively. If the normalized model $G_m(s)$ is defined for the normalized process $G(s)/(4d|G(i\omega)|/\pi)$, then the ultimate amplitude ratio is one and the model for the output nonlinear static function is $z(t) = \hat{f}_1 y_k + \hat{f}_2 y_k^2 + \hat{f}_3 y_k^3 + \dots + \hat{f}_n y_k^n$. Because the overall input–output relation of the original process is the same as that of the normalized process, the model of $|G_m(i\omega)| = 1$ and $z(t) = \hat{f}_1 y_k + \hat{f}_2 y_k^2 + \hat{f}_3 y_k^3 + \dots + \hat{f}_n y_k^n$ obtained is acceptable. The output nonlinear static models of $\hat{z}(t) = 1.417y + 1.146y^2 - 0.020y^3$ is obtained from (12.9). Figure 12.16 shows the performances of the model. It is remarkable that the nonlinear process with the output nonlinearity and static disturbance can be identified using only one relay test. The MATLAB code to estimate the output nonlinear function is shown in Table 12.8. The code should be executed after the code in Table 12.7 is executed, because it uses the activated process input and output data generated by the code in Table 12.7.

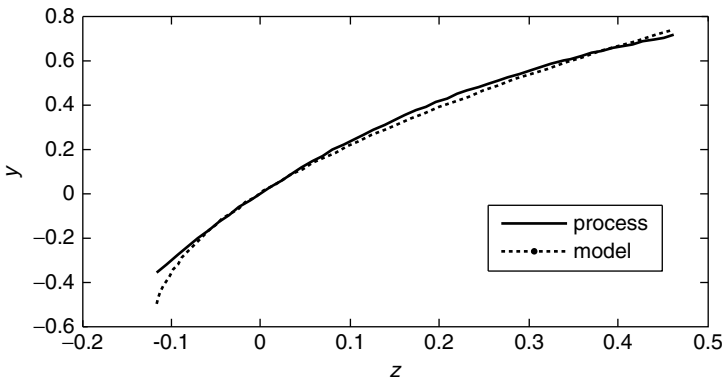


Figure 12.16 Identification results for the output nonlinear static function in Example 12.7.

Table 12.8 MATLAB code to estimate the nonlinear function for the case of a ramp disturbance plus measurement noise in Example 12.7.

```

                                relay_ND3_nonlinear.m
n=length(data_y_u); %data_y_u(:,1:3)=[t y u]
period=data_y_u(n,1)-data_y_u(1,1); w=2*pi/period;
for i=1:n
    matrix(i,1)=-1; matrix(i,2)=data_y_u(i,2);
    matrix(i,3)=data_y_u(i,2)^2; matrix(i,4)=data_y_u(i,2)^3;
    matrix_y(i,1)=sin(w*(i-1)*delt);
end %Estimate the inverse f of y=f(z)
coeff=inv(matrix'*matrix)*matrix'*matrix_y;
fprintf('wu=%5.3f \n', w);
fprintf('f1=%5.3f, f2=%5.3f, f3=%5.3f\n', coeff(2), coeff(3), coeff(4));
% comparison of the model and the process
ymax=max(data_y_u(:,2)); ymin=min(data_y_u(:,2));
s=complex(0,1)*w; giw=exp(-0.5*s)/(s+1)^2; normal=pi/(4*abs(giw)*d);
for i=1:50
    y=ymin+(ymax-ymin)*i/50;
    z=coeff(2)*y+coeff(3)*y^2+coeff(4)*y^3; %model
    mz(i,1)=z/normal; my(i,1)=y; %my=f(mz)
end
figure(3); z=min(mz):(max(mz)-min(mz))/50.0:max(mz);
nn=length(z);
for i=1:nn
    y(i)=1-(1+z(i)/3.0)*exp(-3.0*z(i));
end
plot(z,y,'r'); hold on %real nonlinear function plotting
plot(mz,my); %model nonlinear function plotting

```

```

                                command window
>> relay_ND3_nonlinear
wu=1.893
f1=1.417, f2=1.146, f3=-0.020

```

12.3.1 Concluding Remarks

Relay_ND has been proposed to manipulate output nonlinearities and static disturbances. It guarantees a symmetric relay output by setting the time-length of the lower value of the relay to the half-period. Also, it rejects the effects of static disturbances and output nonlinearity by changing the input reference value of the relay.

12.4 Relay Feedback Method for a Large Range of Operation

This section introduces the two-channel relay feedback method to guarantee a prespecified phase angle of the model under a large range of operation, possibly larger than the magnitude of the relay (Sung *et al.*, 2006). Let us call it Relay_LO. It uses a conventional relay followed by two channels, a proportional channel and an integrator plus relay channel. The phase angle can be specified easily by adjusting the ratio of the two gains of the two channels. It removes the

effects of static disturbances, meaning that one need not define the deviation variables initially, which is very attractive from a practical point of view.

It initially raises the process output up to the reference value y_{ref} using the startup mode and it uses the normal mode for the normal operation. The detailed algorithm is as follows:

$$u_1(t) = 1 \quad \text{for } y(t) < y_{\text{ref}} \quad (12.10)$$

$$u_1(t) = -1 \quad \text{for } y(t) \geq y_{\text{ref}} \quad (12.11)$$

$$s(t) = K_s \int_0^t u_1(\tau) d\tau \quad (12.12)$$

For the startup mode:

$$u_{\text{ref}}(t) = s(t) \quad (12.13)$$

$$u(t) = K_p u_1(t) + u_{\text{ref}}(t) \quad (12.14)$$

For the normal mode:

$$u_2(t) = 1 \quad \text{for } s(t) < s_{\text{ref}}(t) \quad (12.15)$$

$$u_2(t) = -1 \quad \text{for } s(t) \geq s_{\text{ref}}(t) \quad (12.16)$$

$$u_p(t) = K_p u_1(t) \quad (12.17)$$

$$u_i(t) = K_i u_2(t) \quad (12.18)$$

$$u(t) = u_p(t) + u_i(t) + u_{\text{ref}}(t) \quad (12.19)$$

Equations (12.10) – (12.11) and (12.15) – (12.16) are relay 1 and relay 2, respectively. K_s in the startup mode is to adjust the speed of the process input for the process output to reach up to y_{ref} . Note that $u(t)$ in the normal mode is composed of the two channels

$$\text{Ch. 1: } y_{\text{ref}} - y(t) \rightarrow \text{Relay 1} \rightarrow u_1(t) \rightarrow u_p(t)$$

and

$$\text{Ch. 2: } y_{\text{ref}} - y(t) \rightarrow \text{Relay 1} \rightarrow u_1(t) \rightarrow \text{Integrator} \rightarrow s_{\text{ref}}(t) - s(t) \rightarrow \text{Relay 2} \rightarrow u_2(t) \rightarrow u_i(t)$$

$s_{\text{ref}}(t)$ and $u_{\text{ref}}(t)$ are updated each half-cycle of $u_1(t)$ and each cycle of $u_1(t)$ respectively by the following update algorithms:

$$s_{\text{ref},k+1}(t) = \frac{s_{\text{max},k} + s_{\text{min},k}}{2}$$

and

$$u_{\text{ref},k+1}(t) = u_{\text{ref},k}(t) + \frac{\alpha(K_p + K_i)(P_{\text{on},k} - P_{\text{off},k})}{P_{\text{on},k} + P_{\text{off},k}}$$

with the initial value of $u_{\text{ref},1}(t) = (s_{\text{max},1} + s_{\text{min},1})/2$ for $k = 1, 2, 3, \dots$. Here, k denotes the k th cycle of $u_1(t)$. $s_{\text{max},k}$ and $s_{\text{min},k}$ are the maximum value and the minimum value of $s(t)$ during the k th cycle. $P_{\text{on},k}$ and $P_{\text{off},k}$ denote the time-lengths corresponding to the relay on and the relay off respectively of relay 1.

The update rule of u_{ref} makes the process input symmetric (equivalently, the time-average value of the process input is zero) to reject the effects of static disturbances.

$$\frac{(K_p + K_i)(P_{\text{on},k} - P_{\text{off},k})}{P_{\text{on},k} + P_{\text{off},k}}$$

is the time-average value of the process input of the k th cycle. Then, the update rule can be derived:

$$u_{\text{ref},k+1}(t) = u_{\text{ref},k}(t) + \frac{\alpha(K_p + K_i)(P_{\text{on},k} - P_{\text{off},k})}{P_{\text{on},k} + P_{\text{off},k}}$$

$\alpha = 1.5-2.0$ is recommended with a compromise between the convergence rate and the robustness. A larger α increases the convergence rate while it decreases the robustness. The update of $s_{\text{ref},k+1}(t) = (s_{\text{max},k} + s_{\text{min},k})/2$ is to make $u_2(t)$ symmetric.

The describing function for Ch. 1 is

$$N_p(a) = \frac{4K_p}{\pi a}$$

The describing function for Ch. 2 is

$$N_i(a) = K_i \frac{4}{\pi a} \exp\left(\frac{-i\pi}{2}\right) = -iK_i \frac{4}{\pi a}$$

because the integrator of (12.12) shifts the phase as much as $-\pi/2$. Then, the overall describing function of the two-channel relay feedback method is (12.20).

$$N(a) = K_p \frac{4}{\pi a} - iK_i \frac{4}{\pi a} = \frac{4}{\pi a} (K_p - iK_i) \quad (12.20)$$

where a is the peak value of the process output. Then, the identified frequency response model is (12.21) and (12.22):

$$G(i\omega) = -\frac{1}{N(a)} = -\frac{\pi a(K_p + iK_i)}{4(K_p^2 + K_i^2)} \quad (12.21)$$

$$\angle G(i\omega) = -\pi + \arctan\left(\frac{K_i}{K_p}\right) \quad (12.22)$$

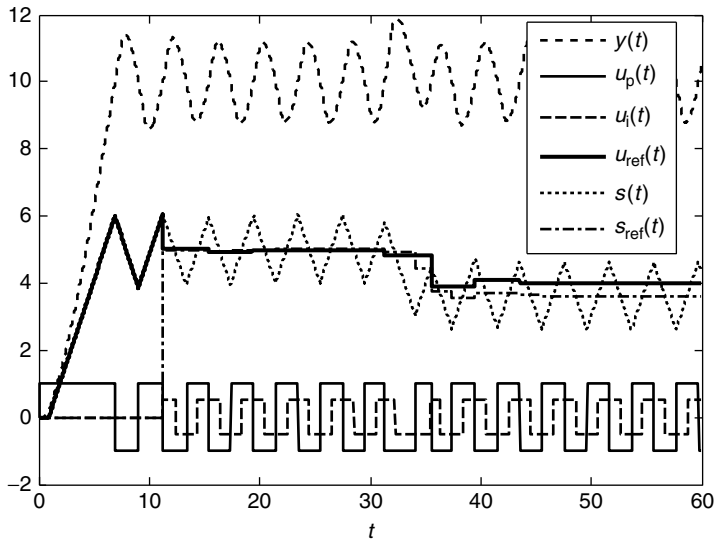


Figure 12.17 Process activation by the two-channel relay feedback method in Example 12.8.

From (12.22), it is clear that the phase angle of the model $\angle G(i\omega)$ can be set by adjusting the ratio of K_i/K_p .

Example 12.8

Simulate the two-channel relay for the following process. The step input disturbance of 1.0 enter at $t = 30.0$.

$$G(s) = 2.0 \frac{\exp(-0.5s)}{(s+1)^2} \quad (12.23)$$

Solution The activated process output is shown in Figure 12.17 for the case of $K_p = 1.5$, $K_i = 0.5$ and $K_s = 1.0$. The MATLAB code to simulate Figure 12.17 and Example 12.8 is shown in Table 12.9.

Table 12.9 MATLAB code to simulate the two-channel relay in Example 12.8.

```

                                relay_LD_ex1.m
clear; tf=60.0; delt=0.01; tf_k=round(tf/delt);
relay1=0.0; relay2=0.0; relay2b=0.0; u=zeros(1,500);
uu=0.0; x=zeros(2,1); y=0.0; yb=0.0;
sum=0.0; sumb=0.0; sumref=0.0;
sum_min=0.0; sum_max=0.0; y_max=-10^10; y_min=10^10;
number=0; yref=10.0; uref=0.0;
kp=1.5; ki=0.5; ks=1.0;
umax=100.0; umin=-100.0; index=0; y_delta=0.2;
for k=1:tf_k

```

Table 12.9 (Continued)

```

    t=(k-1)*delt; T(k)=t; Y(k)=y; U(k)=uu;
    Up(k)=relay1; Ui(k)=relay2; Uref(k)=uref;
    Sum(k)=sum; Sum_ref(k)=sumref;
    for i=1:499 u(i)=u(i+1); end
    if(index==0) %before relay feedback
        relay1=1.0;
        if(y>y_delta)
            index=1; t_off=t; t_on=0.0; Pon=t;
            Poff=0; Period=2*Pon; w=2*pi/Period;
        end
    else
        if(y-yref>=0.0 & yb-yref<0.0)
            sum_max=sum; a_max=y_max; y_max=-10.0^10;
            t_off=t; Pon=t-t_on; P=Pon+Poff; w=2*pi/P;
            number=number+1;
            if(number==2) uref=(sum_max+sum_min)/2; end
            if(number>=2)
                uref=uref+1.5*(kp+ki)*(Pon-Poff)/P;
            end
            if(number>=2) sumref=(sum_max+sum_min)/2; end
        end
        if(y>yref) relay1=-1; if(y>y_max) y_max=y; end; end
        if(y-yref<=0.0 & yb-yref>0.0)
            sum_min=sum; a_min=y_min; y_min=10.0^10;
            if(number>=3) sumref=(sum_max+sum_min)/2; end
            t_on=t; Poff=t-t_off;
        end
        if(y<=yref) relay1=1; if(y<y_min) y_min=y; end; end
        sum=sumb+ks*relay1*delt;
        if(sum>umax) sum=umax; end % saturation
        if(sum<umin) sum=umin; end
        if(sum<=umax & sum>=umin) sumb=sum; end
        if(number<2)
            uref=sum; %relay2=0;
        else
            if(sum>=sumref) relay2=ki; end % square signal
            if(sum<sumref) relay2=-ki; end % square signal
        end
        end
        if(t>30) dis=1.0; else dis=0.0; end
        uu=kp*relay1+relay2+uref; u(500)=uu+dis; yb=y;
        [x,y]=g_relay_LD_ex1(x,delt,u);
        end
        a=(a_max-a_min)/2; giw=-pi*a*(kp+complex(0,1)*ki)/(kp^2+ki^2)/4;
        fprintf('w=%5.3f, giw=(%5.3f)+i(%5.3f) \n',w,real(giw),imag(giw));
        figure(1); plot(T,Y,'b',T,Up,'g',T,Ui,'r',T,Uref,'c',T,Sum,'k',T,
        Sum_ref,'y');

```

Table 12.9 (Continued)

```

                                g_relay_LD_ex1.m
function [next_x,y]=g_relay_LD_ex1(x,delt,u);
subdelt=delt/10; n=round(delt/subdelt);
A=[0 -1 ; 1 -2 ]; B=[2 ; 0];
C=[0 1]; delay=0.5; delay_k=round(delay/delt);
for i=1:n
    dx=A*x+B*u(500-delay_k);
    x=x+dx*subdelt;
end
next_x=x; y=C*x;
return

```

```

                                command window
>> relay_LD_ex1
w=1.555, giw=(-0.565)+i(-0.188)

```

Even though the reference value $y_{\text{ref}} = 10$ is bigger than the process input corresponding to the relay magnitude, the two-channel relay feedback method works properly due to the startup mode. Also, it rejects the effects of the static disturbance effectively. The identified frequency data is close to the real data of $G(i1.555) = -0.546 - i0.209$ and the phase angle $\arctan 2(-0.188, -0.565)$ of the identified frequency model is the same as the prespecified value of $-\pi + \arctan(K_i/K_p)$.

12.4.1 Concluding Remarks

Relay_LO can incorporate a large range of operation using the startup mode. The phase angle of the model can be prespecified by adjusting the ratio K_i/K_p . The effects of static disturbances can be rejected by updating $u_{\text{ref}}(t)$.

Problems

12.1 Find all the processes to which the unbiased-relay feedback method can be applied.

- (a) $G(s) = 1/(10s + 1)$
- (b) $G(s) = 1/(10s + 1)(s + 1)$
- (c) $G(s) = (-5s + 1)/(10s + 1)(s + 1)$
- (d) $G(s) = \exp(-0.05s)/(10s + 1)$
- (e) $G(s) = 2.0\exp(-0.1s)/(3s - 1)(s + 1)$.

12.2 Activate the process $G(s) = 1/(s + 1)^5$ using the unbiased-relay and the biased-relay feedback methods. Estimate the frequency response using the describing function analysis in Chapter 8.

12.3 Activate the process $G(s) = 1/(s + 1)^5$ of which the output is corrupted by uniformly distributed measurement noise between -0.05 and 0.05 using the unbiased-relay and the

biased-relay feedback methods. Estimate the frequency response using the describing function analysis in Chapter 8.

- 12.4 Activate the process $G(s) = 1/(s + 1)^5$ that has a step input disturbance of 0.3 from the beginning using the unbiased-relay feedback method combined with static disturbance rejection technique in Section 12.2. Estimate the frequency response using the describing function analysis in Chapter 8.
- 12.5 Activate the process $G(s) = 1/(s + 1)^5$ at $y(t) = 0.5$ using the unbiased-relay feedback method combined with static disturbance rejection technique in the Section 12.2. Estimate the frequency response using the describing function analysis in Chapter 8.
- 12.6 Activate the following process of using Relay_ND in Section 12.3 and estimate the frequency response using the describing function analysis in Chapter 8. The process has a static disturbance.

$$\frac{d^4 y(t)}{dt^4} + 4 \frac{d^3 y(t)}{dt^3} + 6 \frac{d^2 y(t)}{dt^2} + 4 \frac{dy(t)}{dt} + y(t) = -0.1 \frac{du(t-0.1)}{dt} + u(t-0.1) + d(t)$$

$$\left. \frac{d^3 y(t)}{dt^3} \right|_{t=0} = \left. \frac{d^2 y(t)}{dt^2} \right|_{t=0} = \left. \frac{dy(t)}{dt} \right|_{t=0} = y(0) = 0, \quad u(t) = 0 \quad \text{for } t < 0,$$

$$d(t) = 0.5 \quad \text{for } t \geq 0$$

- 12.7 Activate the following Wiener-type nonlinear process with a step input disturbance using Relay-ND in Section 12.3 and estimate the ultimate frequency response of the linear dynamic subsystem and the output nonlinear static function:

$$w(t) = u(t) + 1.0$$

$$\frac{d^4 z(t)}{dt^4} + 4 \frac{d^3 z(t)}{dt^3} + 6 \frac{d^2 z(t)}{dt^2} + 4 \frac{dz(t)}{dt} + z(t) = -0.1 \frac{dw(t-0.1)}{dt} + w(t-0.1)$$

$$y(t) = 1 - (1 + 2z(t))\exp(-2z(t))$$

- 12.8 Activate the process in Problem 12.6 at $y_{\text{ref}} = 0.5$ using the two-channel relay in Section 12.4 to estimate the frequency responses for the phase angles $-\pi/2$, $-3\pi/4$, $-\pi$.
- 12.9 Activate the virtual process of Process 3 (refer to the Appendix for details) using the following relays and estimate the frequency responses. Also, tune the PID controller using the ZN or IMC tuning rule and show the control performance.
- Unbiased relay, $y_{\text{ref}} = 0.0$; phase angle, $-\pi$.
 - Biased relay, $y_{\text{ref}} = 0.3$; phase angle is not specified.
 - Unbiased relay combined with disturbance rejection technique, $y_{\text{ref}} = 0.3$; phase angle, $-\pi$.
 - Relay_ND, $y_{\text{ref}} = 0.3$; phase angle, $-\pi$.
 - Two-channel relay, $y_{\text{ref}} = 0.3$; phase angle, $-3\pi/4$.

Here, the Fourier analysis or the modified Fourier transform should be used for (b) because the oscillation is asymmetric.

References

- Åström, K.J. and Hägglund, T. (1984) Automatic tuning of simple regulators with specifications on phase and amplitude margins. *Automatica*, **20**, 645.
- Park, J.H., Sung, S.W. and Lee, I. (1997) Improved relay auto-tuning with static load disturbance. *Automatica*, **33**, 711.
- Shen, S., Wu, J. and Yu, C. (1996a) Use of biased-relay feedback for system identification. *AIChE Journal*, **42**, 1174.
- Shen, S., Wu, J. and Yu, C. (1996b) Autotune identification under load disturbance. *Industrial & Engineering Chemistry Research*, **35**, 1642.
- Sung, S.W. and Lee, J. (2006) Relay feedback method under nonlinearity and static disturbance conditions. *Industrial & Engineering Chemistry Research*, **45**, 4028.
- Sung, S.W., Lee, J., Lee, D.H. *et al.* (2006) A two-channel relay feedback method under static disturbances. *Industrial & Engineering Chemistry Research*, **45**, 4071.

13

Modifications of Relay Feedback Methods

13.1 Process Activation Method Using Pulse Signals¹

This section introduces a closed-loop process activation method to reduce the harmonics and obtain more accurate frequency-response data of the process. It can also successfully remove the effect of the input nonlinearity by using the disturbance rejection technique. The method combines 10 pulses to generate one period of the relay signal. The 10 pulses are combined in an optimal way by solving a constrained nonlinear optimization problem to minimize the harmonics. In the implementation, the closed-loop process activation method uses the optimal solution obtained without continuing to solve the optimization problem any more. So, the implementation of the proposed method is almost as simple as that of the previous methods. Let us call it Relay_PS.

The signal generated by Relay_PS (Je *et al.*, 2009) has five pulses in the half-period, as shown in Figure 13.1. The first cycle is activated by the conventional relay feedback method. First, the relay output (equivalently, process input) $u(t) = \alpha d$ is entered until the process output $y(t)$ deviates from the initial value. After that, one cycle is determined as follows: $u(t) = -\alpha d$ when $y(t) \geq 0$ and $u(t) = \alpha d$ when $y(t) < 0$. Here, d is the magnitude of the multi-pulse signal and $\alpha < 1$ is introduced for a smooth transit from the conventional relay mode to the multi-pulse mode without changing the fundamental frequency term, which will be explained later. The multi-pulse signal of Relay_PS begins to enter after one cycle of the conventional relay signal, as shown in Figure 13.1. The process input of the k th cycle in the multi-pulse mode is determined as follows.

When $y(t) < 0$, the following rules are applied, as shown in Figure 13.1:

$$u(t) = d, \quad 0 \leq t - t_{\text{on},k} < x(1)P_{\text{on},k-1} \quad (13.1)$$

$$u(t) = 0, \quad x(1)P_{\text{on},k-1} \leq t - t_{\text{on},k} < x(2)P_{\text{on},k-1} \quad (13.2)$$

¹ Enhanced process activation method to remove harmonics and input nonlinearity, Je *et al.* *Journal of Process Control* Copyright ©[2008] Elsevier, Inc.

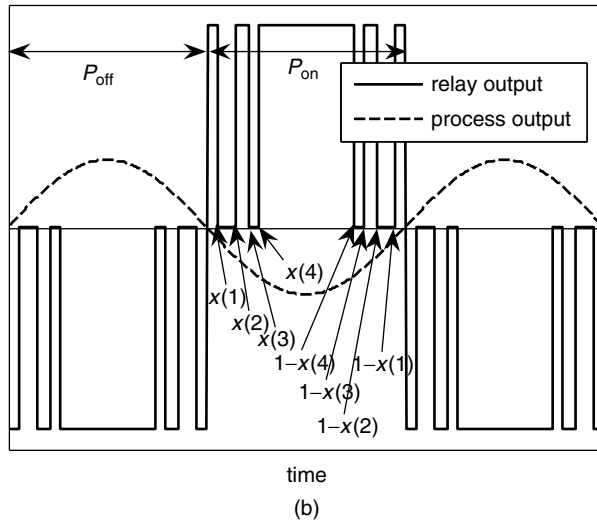
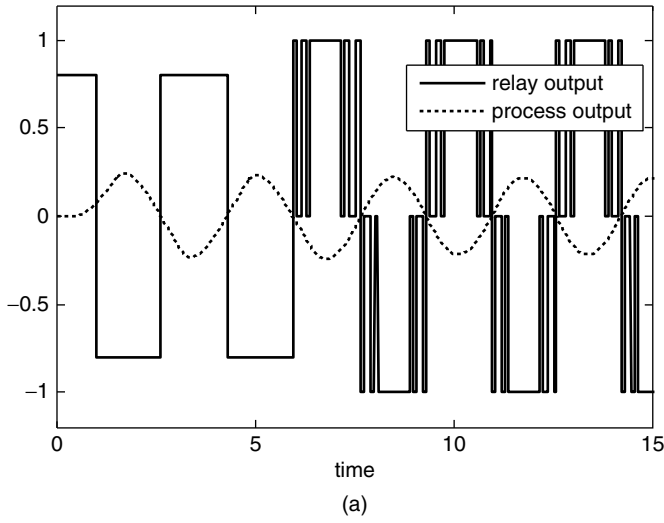


Figure 13.1 Output signal of Relay_PS (a) and its cyclic-steady-state part when the period is unity (b). Enhanced process activation method to remove harmonics and input nonlinearity, Je *et al.* *Journal of Process Control* Copyright ©[2008] Elsevier, Inc.

$$u(t) = d, \quad x(2)P_{\text{on},k-1} \leq t - t_{\text{on},k} < x(3)P_{\text{on},k-1} \quad (13.3)$$

$$u(t) = 0, \quad x(3)P_{\text{on},k-1} \leq t - t_{\text{on},k} < x(4)P_{\text{on},k-1} \quad (13.4)$$

$$u(t) = d, \quad x(4)P_{\text{on},k-1} \leq t - t_{\text{on},k} < (1 - x(4))P_{\text{on},k-1} \quad (13.5)$$

$$u(t) = 0, \quad (1 - x(4))P_{\text{on},k-1} \leq t - t_{\text{on},k} < (1 - x(3))P_{\text{on},k-1} \quad (13.6)$$

$$u(t) = d, \quad (1 - x(3))P_{\text{on},k-1} \leq t - t_{\text{on},k} < (1 - x(2))P_{\text{on},k-1} \quad (13.7)$$

$$u(t) = 0, \quad (1 - x(2))P_{\text{on},k-1} \leq t - t_{\text{on},k} < (1 - x(1))P_{\text{on},k-1} \quad (13.8)$$

$$u(t) = d, \quad (1 - x(1))P_{\text{on},k-1} \leq t - t_{\text{on},k} \quad (13.9)$$

When $y(t) \geq 0$, the following rules are applied, as shown in Figure 13.1:

$$u(t) = -d, \quad 0 \leq t - t_{\text{off},k} < x(1)P_{\text{off},k-1} \quad (13.10)$$

$$u(t) = 0, \quad x(1)P_{\text{off},k-1} \leq t - t_{\text{off},k} < x(2)P_{\text{off},k-1} \quad (13.11)$$

$$u(t) = -d, \quad x(2)P_{\text{off},k-1} \leq t - t_{\text{off},k} < x(3)P_{\text{off},k-1} \quad (13.12)$$

$$u(t) = 0, \quad x(3)P_{\text{off},k-1} \leq t - t_{\text{off},k} < x(4)P_{\text{off},k-1} \quad (13.13)$$

$$u(t) = -d, \quad x(4)P_{\text{off},k-1} \leq t - t_{\text{off},k} < (1 - x(4))P_{\text{off},k-1} \quad (13.14)$$

$$u(t) = 0, \quad (1 - x(4))P_{\text{off},k-1} \leq t - t_{\text{off},k} < (1 - x(3))P_{\text{off},k-1} \quad (13.15)$$

$$u(t) = -d, \quad (1 - x(3))P_{\text{off},k-1} \leq t - t_{\text{off},k} < (1 - x(2))P_{\text{off},k-1} \quad (13.16)$$

$$u(t) = 0, \quad (1 - x(2))P_{\text{off},k-1} \leq t - t_{\text{off},k} < (1 - x(1))P_{\text{off},k-1} \quad (13.17)$$

$$u(t) = -d, \quad (1 - x(1))P_{\text{off},k-1} \leq t - t_{\text{off},k} \quad (13.18)$$

where $P_{\text{on},k-1}$ and $P_{\text{off},k-1}$ denote the time-length of the half-cycle corresponding to the on status and the off status respectively of the $(k-1)$ -th cycle in Relay_PS. $t_{\text{on},k}$ and $t_{\text{off},k}$ represent the starting time of the on status and the off status in the k th cycle respectively. This procedure is repeated until the time-length of the cycles converges.

The x values in (13.1)–(13.18) are determined in an optimal way by solving the constrained nonlinear optimization problem (13.19)–(13.21) to minimize the harmonics. As in (13.19), 11 frequency terms are considered. Frequency terms higher than the 11th term will be negligible because the process dynamics dissipate their effects. If one wants to consider more frequency terms, then one can just replace 11 by a larger value, but it has no special meanings for the usual processes.

$$\min_x \left[\sum_{n=2}^{11} \left(\frac{b_n}{b_1} \right)^2 \right] \quad (13.19)$$

$$b_n = \frac{2}{\pi} \int_0^\pi u(t) \sin(nt) dt = \frac{2}{\pi} \left[\int_0^{x(1)\pi} \sin(nt) dt + \int_{x(2)\pi}^{x(3)\pi} \sin(nt) dt + \int_{x(4)\pi}^{\pi - x(4)\pi} \sin(nt) dt + \int_{\pi - x(3)\pi}^{\pi - x(2)\pi} \sin(nt) dt + \int_{\pi - x(1)\pi}^\pi \sin(nt) dt \right] \quad (13.20)$$

subject to

$$x(1) \geq 0.05, \quad x(2) - x(1) \geq 0.05, \quad x(3) - x(2) \geq 0.05, \quad x(4) - x(3) \geq 0.05, \quad x(4) < 0.5 \quad (13.21)$$

where $u(t)$ is defined as the combination of the 10 pulse signals, as shown in Figure 13.1, and $d = 1$ is assumed without loss of generality. The process input of the cyclic steady state can be represented by the Fourier series $u(t) = \sum_{n=1}^{\infty} b_n \sin(nt)$ assuming that the period is 2π without loss of generality. The cosine series are not considered because $u(t)$ is an odd function.

The inequality constraints of (13.21) are needed to incorporate practical problems of the actuator. If the width of the pulse signal is chosen as too small a value with the intention to reduce the harmonics sufficiently, then the actuator cannot realize the abrupt change of the signal. For example, the width of the pulse can be constrained to be larger than 5% of the half-period. The setting is just an example. If the dynamics of the actuator are faster, then a smaller value than 0.05 can be chosen, and vice versa.

The optimal solutions of $x(1) = 0.0500$, $x(2) = 0.1454$, $x(3) = 0.2114$ and $x(4) = 0.2614$ are obtained by the `fmincon` function in the optimization toolbox of MATLAB. The MATLAB code for the optimization is shown in Table 13.1.

The fundamental frequency term of the optimal solution is $b_1 = 1.0232d$. Table 13.2 shows the harmonics of Relay_PS and the conventional relay feedback method. Relay_PS shows significantly smaller harmonics. It should also be noted that the frequent switching during one period does not produce any big high-frequency harmonics, as shown in Table 13.2, because they are included in the optimization step. The harmonics of an extremely high frequency can be negligible because the process dynamics exponentially reduce the magnitude. The simulation study in the next section will demonstrate that there are no problems related to the frequent switching.

Because the fundamental frequency term of the conventional relay is $4d/\pi$ and that of Relay_PS is $1.0232d$, it is clear that the value of α in Figure 13.1 should be $1.0232\pi/4$ for the smooth transit from the conventional relay feedback mode to the multi-pulse mode of Relay_PS without changing the fundamental frequency term.

It is possible to increase the number of pulse signals to reduce the harmonic terms further. Also, the number of the pulse signals can be reduced or the inequality constraints of (13.21) can be changed to lighten the load of the actuator due to the frequent switching. But this study confines its approach to this point, since this kind of extension is straightforward.

It should be noted that the above-mentioned optimization problem is not solved further in the implementation. The optimal solution obtained of $x(1) = 0.0500$, $x(2) = 0.1454$, $x(3) = 0.2114$ and $x(4) = 0.2614$ is directly used in the implementation step. Therefore, the implementation steps from (13.1) to (13.18) are quite simple and can be programmed in a very cheap microprocessor.

Table 13.1 MATLAB code to solve the nonlinear optimization problem of (13.19)–(13.21).

<pre> optim.m options = optimset('Display','iter','TolFun',1e-10,'TolX',1e-10); [x fval]=fmincon(@objective,[0.06 0.14 0.21 0.3],[],[],[],[],[],[],[], @mycon,options); for i=1:11 h(i)=harmonics(i,x); end fprintf('x(1)=%5.4f,x(2)=%5.4f,x(3)=%5.4f,x(4)=%5.4f\n',x(1), x(2),x(3),x(4)) fprintf('b=%5.4f,%5.4f,%5.4f,%5.4f,%5.4f,%5.4f,%5.4f,%5.4f,%5.4f, %5.4f,%5.4f\n',h) </pre>	
<pre> mycon.m function [c,ceq] = mycon(x) c(1) = -x(1) + 0.05; c(2) = -x(2) + x(1) + 0.05; c(3) = -x(3) + x(2) + 0.05; c(4) = -x(4) + x(3) + 0.05; c(5) = x(4) - 0.5; ceq = []; return </pre>	<pre> objective.m function [V]=objective(x) for i=1:11 h=harmonics(i,x); coeff(i)=h; end s=0.0; for i=2:11 s=s+(coeff(i)/coeff(1))^2; end V=s; </pre>
<pre> command window >>optim x(1)=0.0500,x(2)=0.1454,x(3)=0.2114,x(4)=0.2614 b=1.0232, 0.0000, -0.0263, 0.0000, 0.0134, 0.0000, 0.0857, 0.0000, - 0.0323, -0.0000, 0.0023 </pre>	
<pre> harmonics.m function [h]=harmonics(n,x) syms f t; f=sin(n*t); g=(2/pi)*(int(f,'t',0,x(1)*pi)+int(f,'t',x(2)*pi,x(3)*pi)+int(f,'t', x(4)*pi,(1-x(4))*pi)+int(f,'t',(1-x(3))*pi,(1-x(2))*pi)+int(f,'t', (1-x(1))*pi,pi)); h=double(g); </pre>	

The following equations can be derived by neglecting the harmonic terms to estimate the ultimate information of the process from the relay feedback test using describing function analysis:

$$\omega_u = \frac{2\pi}{P} \quad (13.22)$$

$$k_{cu} = \frac{b_1}{a} \quad (13.23)$$

where a and P are the peak value of the process output and the period in the cyclic steady state respectively. ω_u and k_{cu} respectively represent the ultimate frequency and the ultimate gain

Table 13.2 Harmonics of Relay_PS and the conventional relay feedback method. Enhanced process activation method to remove harmonics and input nonlinearity, Je *et al. Journal of Process Control* Copyright ©[2008] Elsevier, Inc.

Relay_PS ($b_1 = 1.0232d$)		Conventional method ($b_1 = 4d/\pi$)	
$b_2/d = 0.0000$	$b_2/b_1 = 0.0000$	$b_2/d = 0.0000$	$b_2/b_1 = 0.0000$
$b_3/d = -0.0263$	$b_3/b_1 = -0.0257$	$b_3/d = 4\pi/3$	$b_3/b_1 = 0.3333$
$b_4/d = 0.0000$	$b_4/b_1 = 0.0000$	$b_4/d = 0.0000$	$b_4/b_1 = 0.0000$
$b_5/d = 0.0134$	$b_5/b_1 = 0.0131$	$b_5/d = 4\pi/5$	$b_5/b_1 = 0.2000$
$b_6/d = 0.0000$	$b_6/b_1 = 0.0000$	$b_6/d = 0.0000$	$b_6/b_1 = 0.0000$
$b_7/d = 0.0857$	$b_7/b_1 = 0.0838$	$b_7/d = 4\pi/7$	$b_7/b_1 = 0.1429$
$b_8/d = 0.0000$	$b_8/b_1 = 0.0000$	$b_8/d = 0.0000$	$b_8/b_1 = 0.0000$
$b_9/d = 0.0323$	$b_9/b_1 = 0.0316$	$b_9/d = 4\pi/9$	$b_9/b_1 = 0.1111$
$b_{10}/d = 0.0000$	$b_{10}/b_1 = 0.0000$	$b_{10}/d = 0.0000$	$b_{10}/b_1 = 0.0000$
$b_{11}/d = 0.0023$	$b_{11}/b_1 = 0.0022$	$b_{11}/d = 4\pi/11$	$b_{11}/b_1 = 0.0909$
$b_{12}/d = 0.0000$	$b_{12}/b_1 = 0.0000$	$b_{12}/d = 0.0000$	$b_{12}/b_1 = 0.0000$

obtained from Relay_PS test. This describing analysis method is one example to estimate the frequency-response data set from the process data activated by Relay_PS. If Relay_PS is combined with the disturbance rejection techniques and/or other efficient algorithms, such as Fourier analysis and the modified Fourier transform, it is possible to estimate more accurate frequency-response data sets and to obtain many more frequency-response data sets.

The application of Relay_PS to nonlinear system identification will be exemplified in the subsequent two sections.

13.1.1 Comparisons with Previous Approaches

There are the four previous approaches developed to reduce harmonics. They use a multistep signal, saturation-relay signal, sinusoidal signal or preload relay signal (Lee *et al.*, 1995; Sung *et al.*, 1995; Shen *et al.*, 1996; Tan *et al.*, 2006). Meanwhile, Relay_PS uses pulse signals. This is a remarkable advantage of Relay_PS, because it can reject the effects of the input nonlinearity. For example, consider the Hammerstein process of Figure 13.2. If Relay_PS is applied in combination with the previous strategies to reject static disturbances, then the intermediate signal $v(t)$ becomes a symmetric binary signal for which a unit magnitude can be assumed without loss of generality. Then, it is straightforward to obtain the ultimate data of the linear dynamic subsystem. Refer to Sung and Lee 2006 and Park *et al.* 2006 for detailed descriptions, which use a similar principle to identify Hammerstein, Wiener or Hammerstein–Wiener processes. A simple system identification example is given in the next section.

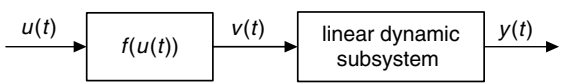


Figure 13.2 Hammerstein-type nonlinear process. Enhanced process activation method to remove harmonics and input nonlinearity, Je *et al. Journal of Process Control* Copyright ©[2008] Elsevier, Inc.

13.1.2 Case Study

Two kinds of simulations were performed. One compares Relay_PS with the conventional relay feedback method. The other demonstrates how the proposed method can be applied to identify the Hammerstein process.

Example 13.1

Several processes are simulated to demonstrate the performances of Relay_PS and the conventional relay feedback method. Tables 13.3–13.5 compares Relay_PS with the conventional relay feedback method for the n th order processes of $G_p(s) = \exp(-\theta s)/(s + 1)^n$, $n = 1, 2, 3$. Table 13.6 shows the estimation results for the minimum-phase-zero process of $G_p(s) = (1 + 0.3s) \exp(-\theta s)/(s + 1)^2$ and the nonminimum-phase-zero process of $G_p(s) = (1 - 0.3s) \exp(-\theta s)/(s + 1)^2$. Table 13.7 indicates the simulation results for the underdamped processes of $G_p(s) = \exp(-0.1s)/(s^2 + 2\xi s + 1)$, $\xi = 0.3, 0.5, 0.7$. For all the cases, Relay_PS provides better estimates than the conventional relay feedback method because it removes the harmonics effectively.

Table 13.3 Simulation results for $G_p(s) = e^{-\theta s}/(s + 1)$.

θ	k_{cu}			ω_u		
	Proposed	Conventional	Process	Proposed	Conventional	Process
0.1	15.0308	13.3722	16.3505	16.3244	16.4481	16.2000
0.5	3.6160	3.2338	3.8069	3.6787	3.7760	3.6733
1.0	2.1667	2.0134	2.2617	2.0249	2.1085	2.0288
5.0	1.0724	1.2818	1.1321	0.5265	0.5521	0.5307

Table 13.4 Simulation results for $G_p(s) = e^{-\theta s}/(s + 1)^2$.

θ	k_{cu}			ω_u		
	Proposed	Conventional	Process	Proposed	Conventional	Process
0.1	20.5552	18.8890	20.6710	4.4373	4.2056	4.4351
0.5	4.6923	4.4239	4.6879	1.9203	1.8982	1.9204
1.0	2.7057	2.5522	2.7069	1.3046	1.3156	1.3065
5.0	1.1611	1.2980	1.2087	0.4588	0.4710	0.4569

Table 13.5 Simulation results for $G_p(s) = e^{-\theta s}/(s + 1)^3$.

θ	k_{cu}			ω_u		
	Proposed	Conventional	Process	Proposed	Conventional	Process
0.1	6.1941	6.0025	6.2164	1.5392	1.5191	1.5430
0.5	3.5422	3.4286	3.5436	1.1544	1.1428	1.1508
1.0	2.4944	2.4146	2.4951	0.9156	0.9189	0.9163
5.0	1.2446	1.1356	1.2494	0.3982	0.4099	0.4000

Table 13.6 Simulation results for $G_p(s) = (1 + \alpha s)e^{-0.3s}/(s + 1)^2$.

α	k_{cu}			ω_u		
	Proposed	Conventional	Process	Proposed	Conventional	Process
-0.3	1.7779	1.7357	1.7792	3.6761	3.3854	3.6750
0.3	4.5399	4.4248	4.5716	12.0737	10.5146	12.9023

Table 13.7 Simulation results for $G_p(s) = e^{-0.1s}/(s^2 + 2\xi s + 1)$.

ξ	k_{cu}			ω_u		
	Proposed	Conventional	Process	Proposed	Conventional	Process
0.3	5.8628	5.2845	6.0692	2.5857	2.4448	2.6196
0.5	9.8006	8.8919	10.1796	3.2057	3.0354	3.2623
0.7	13.8366	12.4866	14.3399	3.7069	3.5102	3.7848

Table 13.8 shows the MATLAB code to simulate Tables 13.3–13.5.

Table 13.8 MATLAB code to simulate Tables 13.3–13.5.

```

                                relay_PS_ex1.m
clear; delt=0.001; tf=30; n=round(tf/delt);
x=zeros(3,1); %table 13.5
%x=zeros(2,1); %table 13.4
%x=zeros(1,1); %table 13.3
u_data=zeros(1,1001);
t_on=0.0; t_off=0.0; P_on=0; P_off=0;
ymin=0.0; ymax=0.0; np=0; y=0.0;
index=0.0; y_delta=0.2;
for i=1:n
    t=i*delt; yy(i)=y; tt(i)=t;
    if(index==0) u=0.8036; if(y>y_delta) index=1; end; end
    if(index==1)
        if(yy(i)>0.0 & yy(i-1)<=0.0)
            P_on=t-t_on; t_off=t;
            ymax_f=ymax; ymax=0.0; end
        if(yy(i)<=0.0 & yy(i-1)>0.0)
            P_off=t-t_off; t_on=t; np=np+1;
            ymin_f=ymin; ymin=0.0; end;
    end
    if(np<=1 & index==1)
        if(y>0.0) u=-0.8036; end
        if(y<=0.0) u=0.8036; end; end
    if(np>1)
        if(y<0.0)
            if(y<ymin) ymin=y; end;
            if(t-t_on<0.05*P_on) u=1;
                elseif(t-t_on<0.1454*P_on) u=0;
                elseif(t-t_on<0.2114*P_on) u=1;

```

Table 13.8 (Continued)

```

elseif (t-t_on<0.2614*P_on) u=0;
elseif (t-t_on<0.7386*P_on) u=1;
elseif (t-t_on<0.7886*P_on) u=0;
elseif (t-t_on<0.8546*P_on) u=1;
elseif (t-t_on<0.95*P_on) u=0;
elseif (0.95*P_on<t-t_on) u=1;
end
end
if (y>=0.0)
    if (y>y_max) y_max=y; end;
    if (t-t_off<0.05*P_off) u=-1;
        elseif (t-t_off<0.1454*P_off) u=0;
        elseif (t-t_off<0.2114*P_off) u=-1;
        elseif (t-t_off<0.2614*P_off) u=0;
        elseif (t-t_off<0.7386*P_off) u=-1;
        elseif (t-t_off<0.7886*P_off) u=0;
        elseif (t-t_off<0.8546*P_off) u=-1;
        elseif (t-t_off<0.95*P_off) u=0;
        elseif (0.95*P_off<t-t_off) u=-1;
    end
end
end
for j=1:1000 u_data(j)=u_data(j+1); end
uu(i)=u; u_data(1001)=u;
[x,y]=process_pwm1(x,delt,u_data);
end
P=P_on+P_off; a=(y_max_f-y_min_f)/2; w_u=2*pi/P
AR_u=a/1.0232; k_cu=1/AR_u;
fprintf('kcu=%5.3f, wu=%5.3f\n',k_cu,w_u);
figure(1); plot(tt,uu,tt,yy);

```

```

                                g_relay_PS_ex1.m
function [next_x,y]=g_relay_PS_ex1(x,delt,u);
subdelt=delt/10; n=round(delt/subdelt);
%table 13.3
%A=-1; B=1; C=1; delay=0.1;
%table 13.4
%A=[0 -1; 1 -2]; B=[1; 0]; C=[0 1]; delay=0.1;
%table 13.5
A=[0 0 -1; 1 0 -3; 0 1 -3];
B=[1; 0; 0]; C=[0 0 1]; delay=0.1;
delay_k=round(delay/delt);
for i=1:n
    dx=A*x+B*u(1000-delay_k);
    x=x+dx*subdelt;
end
next_x=x; y=C*x;
return

```

command window

```

>> relay_PS_ex1
kcu=6.189, wu=1.542

```

Example 13.2

The following Hammerstein process is simulated to demonstrate the additional advantage of Relay_PS compared with the previous approaches developed to reduce harmonics.

$$v(t) = 1 - \exp(-0.3u(t)) \quad (13.24)$$

$$\frac{y(s)}{v(s)} = \frac{\exp(-0.5s)}{(s+1)^2} \quad (13.25)$$

Equation (13.24) is the input nonlinearity and (13.25) is the linear dynamic subsystem. The intermediate signal $v(t)$ is not measurable.

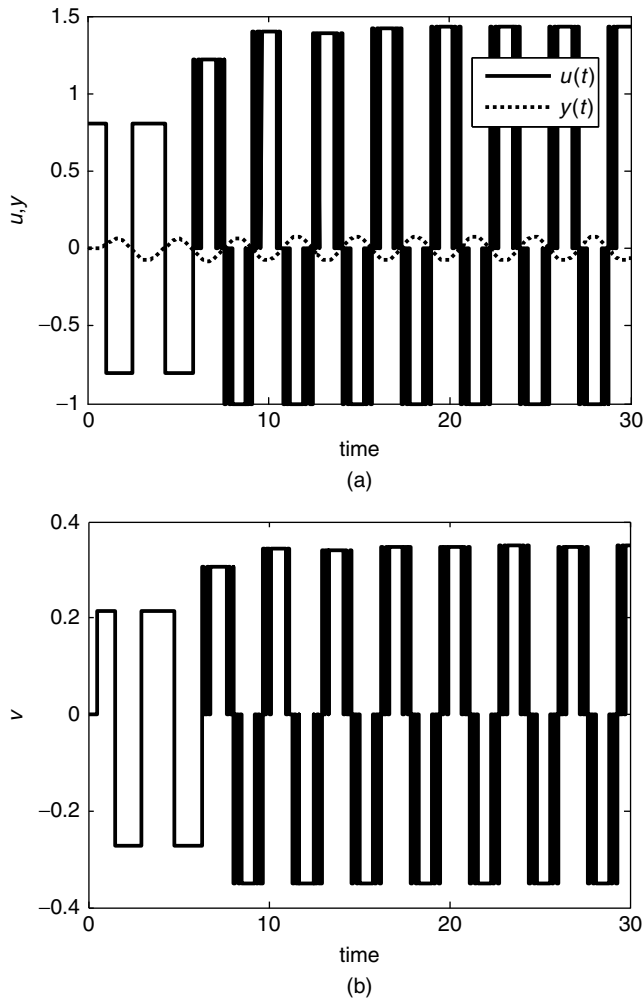


Figure 13.3 Estimation performances of the proposed method for the Hammerstein process: activated process signals (a) and (b); equivalent Hammerstein process (c); modeling result for the input nonlinearity (d). Enhanced process activation method to remove harmonics and input nonlinearity, Je *et al.* *Journal of Process Control* Copyright © [2008] Elsevier, Inc.

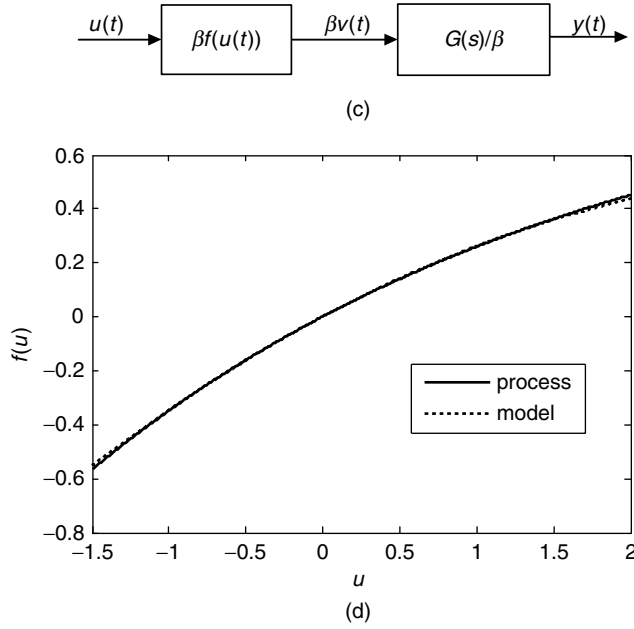


Figure 13.3 (Continued)

Figure 13.3 shows the activated process signals by Relay_PS combined with the disturbance rejection method (Park *et al.*, 1997). The MATLAB code to simulate Figure 13.3a and b is shown in Table 13.9. The upper value of the process input has been updated by the following equation, as shown in Figure 13.3a:

$$u_{\max,k} = u_{\max,k-1} - 2 \left(\frac{a_{\max,k-1} + a_{\min,k-1}}{|a_{\max,k-1}| + |a_{\min,k-1}|} \right) \quad (13.26)$$

where $a_{\max,k-1}$ and $a_{\min,k-1}$ are the $(k-1)$ -th peak value and the $(k-1)$ -th valley value of the process output. $u_{\max,k}$ is the k th upper value of the process input. The update rule (13.26) is just an example. A tuning parameter can be included to compromise between the convergence rate and the robustness. For detailed descriptions on the update rule, refer to Chapter 12.

Note that the update (13.26) results in a symmetric $v(t)$ signal, as shown in Figure 13.3b. Then, it can be assumed that the upper value and the lower value of $v(t)$ are $+1$ and -1 respectively, without loss of generality, because there is one degree of freedom to adjust β , as shown in Figure 13.3c. Then, the ultimate data ($\omega_u = 1.921$ (actual: 1.920) and $k_u = 5.828$ (actual: 4.686)) of the dynamic subsystem $G(s)$ is obtained from the cyclic-steady-state input and output data in Figure 13.3a. Also, the model $\hat{v}(t) = 0.876u(t) - 0.124u(t)^2$ from $0 = \beta f(0)$, $1 = \beta f(1.433)$ and $-1 = \beta f(-1.000)$ is obtained. Here, $\hat{v}(t)$ corresponds to $\beta v(t)$. Figure 13.3d compares the model $\hat{v}(t)/\beta = (0.876u(t) - 0.124u(t)^2)/\beta$ obtained and the actual input nonlinear function of $v(t) = 1 - \exp(-0.3u(t))$. Here, $\beta = 2.863$ is used, estimated by $1/v_{\max}$. It confirms that Relay_PS can successfully identify the Hammerstein process. The previous approaches cannot be applied to this kind of application.

Table 13.10 shows the MATLAB code to estimate the nonlinear model from the process input and output activated by Relay_PS. It should be executed after the code of Table 13.9 is executed because it uses the data obtained by the code of Table 13.9.

13.1.3 Concluding Remarks

A new closed-loop process activation method was introduced to provide more accurate ultimate data estimates by reducing the harmonics as well as the effect of the input nonlinearity. It minimizes the harmonics by combining the ten pulse signals in an optimal way by solving the constrained nonlinear optimization problem. The implementation is quite simple, so that a very cheap microprocessor is enough to realize the proposed algorithm. Simulation examples demonstrate that Relay_PS can remove the harmonics and the input nonlinearity simultaneously.

Table 13.9 MATLAB code to simulate Figure 13.3a and b.

```

                                relay_PS_ex2.m
clear; delt=0.002; tf=30; x=[0;0]; n=round(tf/delt); u_data=zeros
(1,1001);
v=0.0; y=0.0; t_on=0.0; t_off=0.0; P_on=0; P_off=0;
ymin=0.0; ymax=0.0; np=0; umax=1.0; umin=-1.0; index=0.0; y_delta=0.1;
for i=1:n
    t=i*delt; yy(i)=y; tt(i)=t;
    if(index==0) u=0.8036; if(y>y_delta) index=1; end; end
    if(index==1)
        if(yy(i)>0.0 & yy(i-1)<=0.0)
            P_on=t-t_on; t_off=t; ymin_f=ymin; ymax=0.0;
            if(v>0) vmax=v; else vmin=v; end
        end
        if(yy(i)<=0.0 & yy(i-1)>0.0)
            P_off=t-t_off; t_on=t; ymin=0.0; np=np+1; ymax_f=ymax;
            if(v>0) vmax=v; else vmin=v; end
            if(np>=2)
                umax=umax-2.0*((ymax_f+ymin_f)/(abs(ymax_f)+abs(ymin_f)));
            end
        end
    end
    if(np<=1 & index==1)
        if(y>0.0) u=-0.8036; else u=0.8036; end;
    end
    if(y<0.0)
        if(y<ymin) ymin=y; end
    else
        if(y>ymax) ymax=y; end
    end
    if(np>1)
        if(y<0.0)
            if(y<ymin) ymin=y; end
            if(t-t_on<0.05*P_on) u=umax;

```

Table 13.9 (Continued)

```

elseif(t-t_on<0.1454*P_on) u=0; elseif(t-t_on<0.2114*P_on) u=umax;
elseif(t-t_on<0.2614*P_on) u=0; elseif(t-t_on<0.7386*P_on) u=umax;
elseif(t-t_on<0.7886*P_on) u=0; elseif(t-t_on<0.8546*P_on) u=umax;
elseif(t-t_on<0.95*P_on) u=0; elseif(0.95*P_on<t-t_on) u=umax;
end
end
if(y>=0.0)
    if(y>ymax) ymax=y; end
    if(t-t_off<0.05*P_off) u=-1;
    elseif(t-t_off<0.1454*P_off) u=0; elseif(t-t_off<0.2114*P_off)
u=umin;
    elseif(t-t_off<0.2614*P_off) u=0; elseif(t-t_off<0.7386*P_off)
u=umin;
    elseif(t-t_off<0.7886*P_off) u=0; elseif(t-t_off<0.8546*P_off)
u=umin;
    elseif(t-t_off<0.95*P_off) u=0; elseif(0.95*P_off<t-t_off) u=umin;
end
end
end
for j=1:1000 u_data(j)=u_data(j+1); end
vv(i)=v; uu(i)=u; u_data(1001)=u; [x,y,v]=g_relay_PS_ex2(x,delt,
u_data);
end
P=P_on+P_off; a=(ymax_f-ymin_f)/2; wu=2*pi/P; beta=1/vmax; kcu=4/
(a*pi*beta);
fprintf('umax=%5.3f umin=%5.3f vmax=%5.3f vmin=%5.3f\n',umax,umin,
vmax,vmin);
fprintf('beta=%5.3f a=%5.3f wu=%5.3f kcu=%5.3f\n',beta,a,wu,kcu);
figure(1); plot(tt,uu,tt,yy,':'); figure(2); plot(tt,vv);

```

```

                                g_relay_PS_ex2.m
function [next_x,y,v]=g_relay_PS_ex2(x,delt,u);
subdelt=delt/10;
n=round(delt/subdelt);
A=[0 -1; 1 -2]; B=[1; 0];
C=[0 1]; delay=0.5;
delay_k=round(delay/delt);
u_delay=u(1001-delay_k);
v=1-exp(-0.3*u_delay);
for i=1:n
    dx=A*x+B*v;
    x=x+dx*subdelt;
end
next_x=x; y=C*x;
return

```

command window

```

>> relay_PS_ex2
umax=1.432 umin=-1.000 vmax=0.349 vmin=-0.350
beta=2.863 a=0.076 wu=1.919 kcu=5.828

```

Table 13.10 MATLAB code to estimate the nonlinear function.

<pre>relay_PS_ex2_nonlinear.m f=inv([umax umax^2; umin umin^2])*[1;-1]; fprintf('f1=%5.3f f2=%5.3f\n',f(1),f(2)); ui=-1.5; uf=2.0; delu=0.01; n=round((uf-ui)/delu); clear um fm vm for i=1:n um(i)=ui+i*delu; fm(i)=1-exp(-0.3*um(i)); vm(i)=(f(1)*um(i)+f(2)*um(i)^2)/beta; end figure(2); plot(um, fm, um, vm, ':')</pre>	<pre>command window >> relay_PS_ex2_nonlinear f1=0.876 f2=-0.124</pre>
--	--

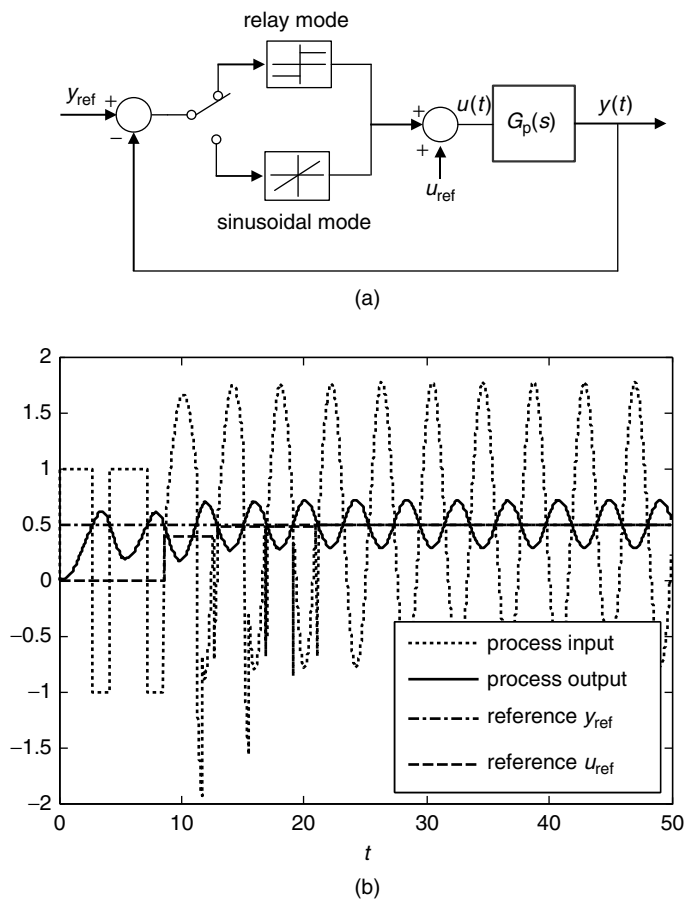


Figure 13.4 Schematic diagram of Relay_SS (a) and typical responses (b). Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

13.2 Process Activation Method Using Sine Signals²

This section introduces the process activation method using sine signals to remove the harmonics of the process input completely, to guarantee the specified phase angle and to reject the effects of disturbances (Sung and Lee, 2008). Also, the stability conditions of the process activation method are discussed. Let us call it Relay_SS.

Figure 13.4 shows the schematic diagram and the typical response of Relay_SS.

Relay_SS uses one cycle of the conventional relay feedback signal followed by a sinusoidal signal combined with a pulse signal. The role of the pulse signal is to enforce the process output and the process input to converge to the complete sinusoidal signal. In Figure 13.4b, the first several cycles of the incomplete sinusoidal signals converge to the complete sinusoidal signal by the guidance of the pulse signals. Then, the important conclusion reached is that the process input and the process output in the cyclic steady state include no harmonics. The reference u_{ref} for the process input is automatically updated to enforce the oscillation of the process input and output to be symmetric by rejecting the effects of the static disturbance. In Figure 13.4b, the reference value for the process output in the relay feedback method is $y_{\text{ref}} = 0.5$, which is equivalent to introducing a static disturbance. Nevertheless, Relay_SS automatically raises the process output up to the reference value of $y_{\text{ref}} = 0.5$ and provides the symmetric oscillation of the complete sinusoidal signal by updating u_{ref} .

Consider the following steps to understand the working of the process activation method in more detail. Step 1, obtain one cycle from $t = 0$ to $t \approx 10$ using the conventional relay feedback method. That is, $u(t) = d$ when $y(t) < y_{\text{ref}}$, $u(t) = -d$ when $y(t) \geq y_{\text{ref}}$, where d , $u(t)$, $y(t)$ and y_{ref} denote the magnitude of the relay, the process input, the process output and the reference value of the relay feedback respectively. The role of Step 1 is to initialize the initial parameters of the sinusoidal mode. Step 2, change from the relay feedback mode to the sinusoidal mode. In the sinusoidal mode, Relay_SS determines the process input as follows.

When $y(t) \geq y_{\text{ref}}$ (negative input status):

$$u(t) = -\frac{4d}{\pi} \sin[\omega_{k-1}^{\text{off}}(t - t_{\text{off},k}) - \phi] + u_{\text{ref}} \quad \text{if } t \leq t_{\text{on}}^{\text{expected}} \quad (13.27)$$

$$u(t) = -\frac{4d}{\pi} \sin(\omega_{k-1}^{\text{off}} P_{\text{off},k-1} - \phi) + u_{\text{ref}} \quad \text{if } t_{\text{on}}^{\text{expected}} < t \leq t_{\text{on}}^{\text{expected}} + \delta \quad (13.28)$$

$$u(t) = -\frac{4d}{\pi} \sin(\omega_{k-1}^{\text{off}} P_{\text{off},k-1} - \phi) - \frac{4d}{\pi} + u_{\text{ref}} \quad \text{if } t > t_{\text{on}}^{\text{expected}} + \delta \quad (13.29)$$

$$u(t) = -\frac{4d}{\pi} \sin[\omega_{k-1}^{\text{off}}(t + t_{\text{off},k}) - \phi] - \frac{4d}{\pi} + u_{\text{ref}} \quad \text{if } t \leq t_{\text{off}}^{\text{expected}} - \delta \quad (13.30)$$

² Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

When $y(t) < y_{\text{ref}}$ (positive input status):

$$u(t) = \frac{4d}{\pi} \sin[\omega_{k-1}^{\text{on}}(t - t_{\text{on},k}) - \phi] + u_{\text{ref}} \quad \text{if } t \leq t_{\text{off}}^{\text{expected}} \quad (13.31)$$

$$u(t) = \frac{4d}{\pi} \sin(\omega_{k-1}^{\text{on}} P_{\text{on},k-1} - \phi) + u_{\text{ref}} \quad \text{if } t_{\text{off}}^{\text{expected}} < t \leq t_{\text{off}}^{\text{expected}} + \delta \quad (13.32)$$

$$u(t) = \frac{4d}{\pi} \sin(\omega_{k-1}^{\text{on}} P_{\text{on},k-1} - \phi) + \frac{4d}{\pi} + u_{\text{ref}} \quad \text{if } t > t_{\text{off}}^{\text{expected}} + \delta \quad (13.33)$$

$$u(t) = \frac{4d}{\pi} \sin[\omega_{k-1}^{\text{on}}(t - t_{\text{on},k}) - \phi] + \frac{4d}{\pi} + u_{\text{ref}} \quad \text{if } t \leq t_{\text{on}}^{\text{expected}} - \delta \quad (13.34)$$

where $\omega_{k-1}^{\text{off}} = \pi/P_{\text{off},k-1}$, $\omega_{k-1}^{\text{on}} = \pi/P_{\text{on},k-1}$, $t_{\text{on}}^{\text{expected}} = t_{\text{off},k} + P_{\text{off},k-1}$ and $t_{\text{off}}^{\text{expected}} = t_{\text{on},k} + P_{\text{on},k-1}$. $P_{\text{off},k-1}$ and $P_{\text{on},k-1}$ denote the time-length of the half-cycle in the previous $(k-1)$ -th cycle corresponding to the negative input status and the positive input status respectively. $t_{\text{on},k}$ is defined as the time at which the process input changes from the negative input status to the positive input status, which is the start of the on status of the k th cycle. The negative input status of the k th cycle starts at $t_{\text{off},k}$. $t_{\text{on}}^{\text{expected}} = t_{\text{off},k} + P_{\text{off},k-1}$ and $t_{\text{off}}^{\text{expected}} = t_{\text{on},k} + P_{\text{on},k-1}$ are the expected subsequent t_{on} and t_{off} on the basis of the previous half-periods of $P_{\text{off},k-1}$ and $P_{\text{on},k-1}$. $-\pi + \phi$ corresponds to the prespecified phase angle between the process input and the process output. δ in (13.29), (13.30), (13.33) or (13.34) is a small positive value to prevent undesirable activation due to uncertainties. Because the magnitude of the fundamental frequency term of the relay mode is $4d/\pi$, we choose the magnitude of the sinusoidal signal of $4d/\pi$ for the smoothing transit from the conventional relay mode to the sinusoidal mode without changing the magnitude of the fundamental frequency term.

Let us explain the role of each equation from (13.27) to (13.34). If the process reaches a cyclic steady state (that is, $P_{\text{off},k-1} = P_{\text{off},k}$ and $P_{\text{on},k-1} = P_{\text{on},k}$), then only (13.27) and (13.31) will be activated. It is clear that $u(t)$ of (13.27) or (13.31) has no harmonics and the phase angle between $u(t)$ and $y(t)$ is exactly $-\pi + \phi$ in the cyclic steady state. Then, it is straightforward to estimate the exact frequency model, of which the phase angle is exactly $-\pi + \phi$.

When the continuous cycling of the process input/output deviates from the desired cyclic steady state, Relay_SS forces the continuous cycling to go back to the desired position by adding the additional step signals of $4d/\pi$ to the sinusoidal signals, as shown in (13.29), (13.30) and (13.33), (13.34). For simplicity, assume $\delta = 0$ and consider the mechanisms shown in Figure 13.5.

If $y(t) \geq y_{\text{ref}}$ and $t > t_{\text{on}}^{\text{expected}}$ (that is, the process output has not crossed the reference value yet, although it is expected to cross at $t = t_{\text{on}}^{\text{expected}}$), then the time-length corresponding to the positive status of the process output should be shortened. This can be achieved by simply adding a negative additional signal of $-4d/\pi$ (which corresponds to the negative pulse in Figure 13.5a) to the process input, as shown in (13.29) and Figure 13.5a. Also, the time-length of the negative status of the process output can be shortened by adding the positive additional signal of $4d/\pi$ (which corresponds to the positive pulse in Figure 13.5b) to the process input,

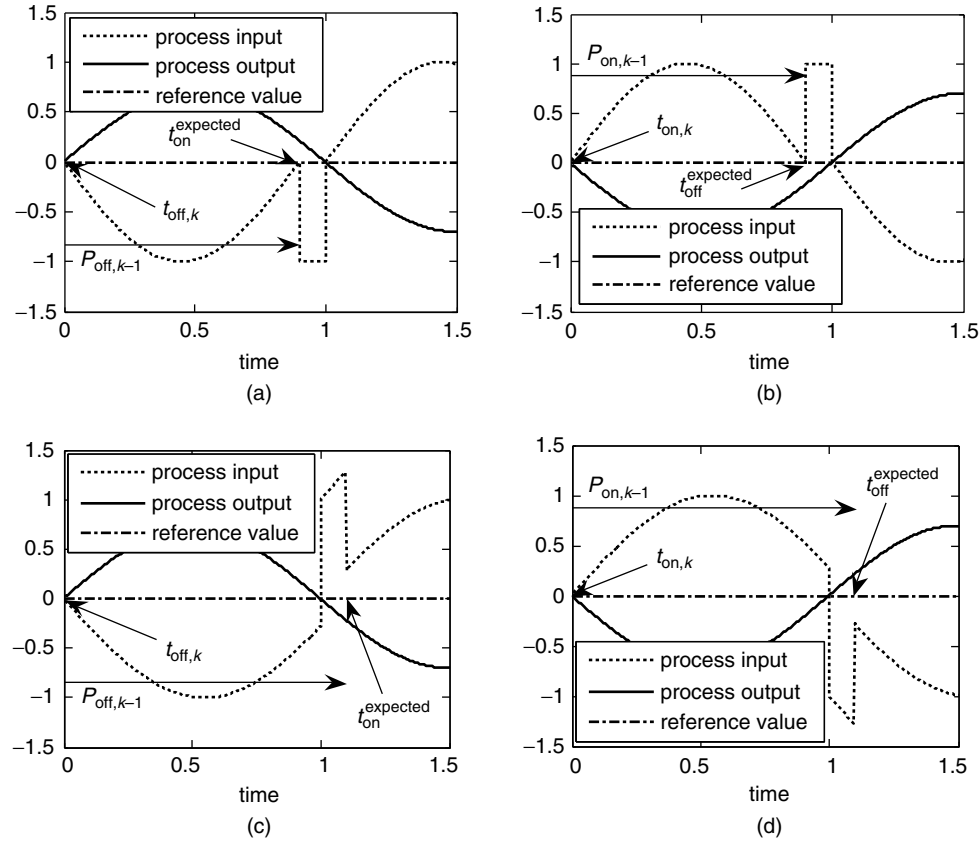


Figure 13.5 Mechanisms to achieve a cyclic steady-state. Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

as shown in (13.33) and Figure 13.5b. This is the same principle as used in the conventional relay feedback method, which shortens the time-length of the positive status of the process output cycle by entering a negative process input, and vice versa.

If $y(t) < y_{ref}$ and $t \leq t_{on}^{expected}$ (that is, the process output crosses the reference value earlier than expected), then the time-length corresponding to the positive status of the process output will be lengthened by adding a positive additional signal of $4d/\pi$ (which corresponds to the positive pulse in Figure 13.5c) to the process input, as shown in (13.34) and Figure 13.5c. The same mechanism is applied to the opposite case, as shown in Figure 13.5d.

Equations (13.28) and (13.32) of the proposed method are to prevent undesirable activation of (13.29), (13.30) and (13.33), (13.34) due to cycle-to-cycle uncertainties, such as measurement noise and high-frequency disturbances. δ will be a small positive value. If the overload of the actuator is of no concern, then (13.28) and (13.32) can be removed.

u_{ref} is updated in each cycle by $u_{ref}(k) = u_{ref}(k-1) + \alpha(2d/\pi)(P_{on}(k-1) - P_{off}(k-1))/P(k-1)$, which shifts u_{ref} as much as the magnitude of the zero-frequency quantity to reject

the effects of the disturbance and to obtain a symmetric input signal. Here, $P_{\text{on}}(k-1)$ and $P_{\text{off}}(k-1)$ are the time-lengths corresponding to the positive status and the negative status of the process input signal respectively. $P(k-1) = P_{\text{on}}(k-1) + P_{\text{off}}(k-1)$ corresponds to the time-length of the $(k-1)$ -th cycle and $(2d/\pi)(P_{\text{on}}(k-1) - P_{\text{off}}(k-1))/P(k-1)$ is the time-average value (equivalently, the zero-frequency quantity) of the previous cycle. Note that $(2d/\pi)(P_{\text{on}}(k-1) - P_{\text{off}}(k-1))/P(k-1)$ monotonically decreases as the process input $u_{\text{ref}} + D$ increases, where D is the input static disturbance. So, the update $u_{\text{ref}}(k) - u_{\text{ref}}(k-1) = \alpha(2d/\pi)(P_{\text{on}}(k-1) - P_{\text{off}}(k-1))/P(k-1)$ will remove the effect of the static input disturbance (equivalently, making $u_{\text{ref}} + D = 0$), resulting in symmetric responses of the process input and output. The convergence of the update method will be discussed in the next section.

In this research, it is recommended that $\alpha = 1.5 \exp[-(n_c - 1)/\tau_\alpha]$. Here, n_c is the number of the sinusoidal cycles. τ_α determines the convergence rate of u_{ref} . A small τ_α means a fast convergence, but too small a τ_α will not provide enough time for u_{ref} to converge to the right value. Therefore, $3 \leq \tau_\alpha \leq 5$ is recommended. Roughly speaking, this requires four to six cycles to obtain symmetric process input and output. A typical performance is shown in Figure 13.4b with $\tau_\alpha = 3$. Here, the reference value of the process output is 0.5. It confirms Relay_SS provides the symmetric cycle for the nonzero reference value. Previous relay feedback approaches cannot provide the symmetric cycle for the nonzero reference value if they have no mechanism to reject static disturbances.

From the activated process data by Relay_SS, the frequency data can be estimated in a straightforward manner. Because the process output in the cyclic steady state is $y(t) = a \sin(\omega t) + y_{\text{ref}}$ for the process input $u(t) = 4d \sin[\omega t - (-\pi + \phi)]/\pi + u_{\text{ref}}$, the identified frequency data of the process is

$$|G(i\omega)| = \frac{\pi a}{4d}, \quad \angle G(i\omega) = -\pi + \phi \quad (13.35)$$

where a is the measured oscillation magnitude of the process output.

Relay_SS removes the effects of static disturbances and provides the oscillation corresponding to the prespecified phase angle. Also, it can remove harmonics completely, resulting in exact frequency-response estimates of (13.35).

13.2.1 Discussions on Convergence

There are insurmountable limitations in proving the convergence in a rigorous way for all the possible situations in operating Relay_SS. So, discussion of the convergence of Relay_SS will be confined to the two restricted situations of Case 1 and Case 2 with several assumptions.

Case 1 Assume that there is no entry of static disturbances (no update of u_{ref}) and that a small perturbation is added to the amplitude or frequency of the signal during the cyclic-steady-state operation. The necessary condition (Loeb condition) for a stable limit cycle in Case 1 is (Atherton, 1975)

$$\left(\frac{\partial U}{\partial \omega} \frac{\partial Q}{\partial a} - \frac{\partial V}{\partial \omega} \frac{\partial P}{\partial a} \right) \bigg|_{\omega=\omega_c} > 0 \quad (13.36)$$

where $G(i\omega) = U(\omega) + iV(\omega)$ and $-1/N(a) = P(a) + iQ(a)$. ω_c is the frequency of the cycle. $N(a)$ is the describing function of Relay_SS. It is clear that $N(a) = (4d/\pi a) \exp(-i\phi)$ because it generates the output $u(t) = (4d/\pi) \sin(\omega t - \phi)$ for the input $-y(t) = a \sin(\omega t)$ and the effects of the additional signals are negligible for such a small perturbation. Then, $P(a) = -(\pi a/4d) \cos(\phi)$ and $Q(a) = -(\pi a/4d) \sin(\phi)$. Then, (13.36) becomes

$$\left(-\frac{\partial U}{\partial \omega} \frac{\pi \sin(\phi)}{4d} + \frac{\partial V}{\partial \omega} \frac{\pi \cos(\phi)}{4d} \right) \bigg|_{\omega=\omega_c} > 0 \quad (13.37)$$

We know that $\cos(\phi)$ and $\sin(\phi)$ are always positive because $0 \leq \phi \leq \pi/2$. Note that most processes for $0 \leq \phi \leq \pi/2$ (equivalently, $-\pi/2 \leq \angle G(i\omega) \leq -\pi$) are one of the two cases $\partial U/\partial \omega|_{\omega=\omega_c} \leq 0$, $\partial V/\partial \omega|_{\omega=\omega_c} > 0$ and $\partial U/\partial \omega|_{\omega=\omega_c} > 0$, $\partial V/\partial \omega|_{\omega=\omega_c} > 0$ because the imaginary part V of most processes increases as ω increases. It is clear that the first case of $\partial U/\partial \omega|_{\omega=\omega_c} \leq 0$ and $\partial V/\partial \omega|_{\omega=\omega_c} > 0$ satisfies (13.37).

Equation (13.37) can be rewritten for the second case of $\partial U/\partial \omega|_{\omega=\omega_c} > 0$ and $\partial V/\partial \omega|_{\omega=\omega_c} > 0$:

$$\tan(\phi) < \frac{\partial V/\partial \omega|_{\omega=\omega_c}}{\partial U/\partial \omega|_{\omega=\omega_c}} \quad (13.38)$$

Consider Figure 13.6. Here, $\tan(\varphi) = (\partial V/\partial \omega|_{\omega=\omega_c})/(\partial U/\partial \omega|_{\omega=\omega_c})$ and $\tan(\phi) = U(\omega_c)/V(\omega_c)$. Most processes satisfy $\phi < \varphi$ for a small ϕ , and equivalently (13.38). Now, it is proven that the proposed test signal generator in Case 1 provides a stable limit cycle for most processes.

Case 2 Assume the situation that a static disturbance enters during the cyclic-steady-state operation and that u_{ref} is updated. The update method can be expressed as in Figure 13.7.

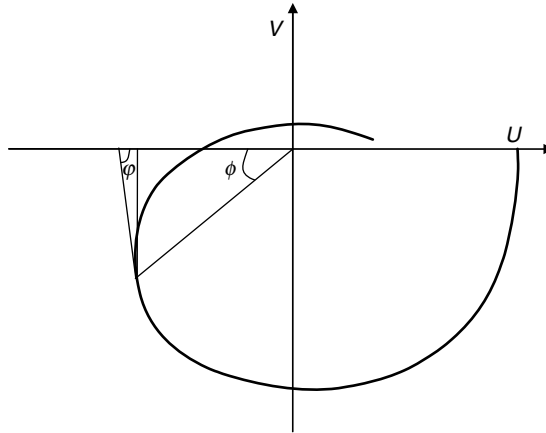


Figure 13.6 $\varphi > \phi$ for most processes. Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

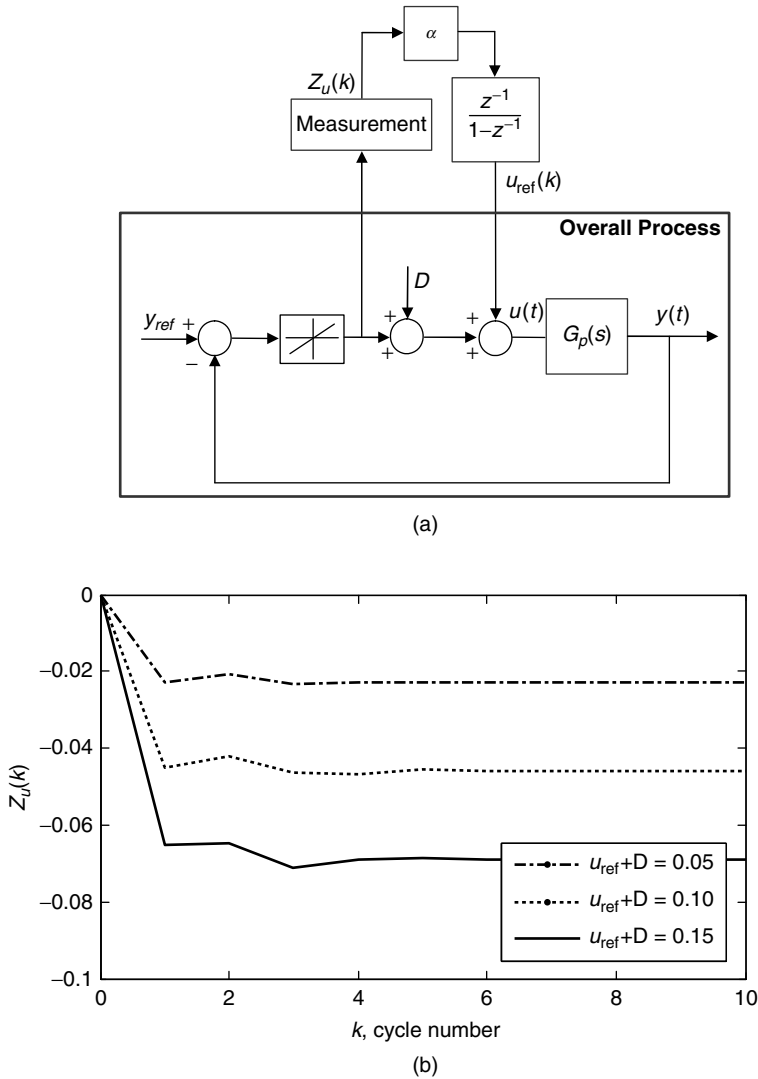


Figure 13.7 u_{ref} update method (a) and a typical step response (b). Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

D denotes the static input disturbance. The zero-frequency quantity of the process input $Z_u(k-1) = (2d/\pi)(P_{on}(k-1) - P_{off}(k-1))/P(k-1)$ can be measured from the previous cycle. Then, the discrete-time overall process can be defined, of which input and output are $u_{ref}(k)$ (or $u_{ref}(k) + D$) and $Z_u(k)$ respectively, as shown in Figure 13.7a. Here, assume that the perturbation by the static disturbance is so small that the internal feedback loop inside the

overall process is internally stable as in Case 1. Now, note that $Z_u(k)$ is proportional to $-(u_{\text{ref}}(k) + D)$ around $u_{\text{ref}}(k) + D = 0$ because the symmetricalness of the signal continuously degrades as $u_{\text{ref}}(k) + D$ is increased. For example, Figure 13.7b shows a typical step response of $Z_u(k)$ for the positive step input of $u_{\text{ref}}(k) + D$. As expected, it shows a linearity between $Z_u(k)$ and $u_{\text{ref}}(k) + D$. Now, it can be inferred that the update method would be more stable as the gain α is decreased.

Let us derive a rough range of α to stabilize the update system. Note that the step response in Figure 13.7b reaches the steady-state within almost one cycle. This is coincident with the fact that relay feedback methods can usually obtain a cyclic-steady-state within such a small number of cycles (2–3 cycles). Then, $Z_u(k)$ is approximately a function of $u_{\text{ref}}(k-1) + D$. Then, a qualitative condition for convergence can be derived as discussed below with the assumption that the approximation is valid.

$Z_y = k_p(D + u_{\text{ref}} + Z_u)$ is valid at the steady state, where Z_y and k_p are the zero-frequency quantity of the process output and the static gain of the process respectively. Assume $k_p > 0$ without loss of generality. Then, $Z_u = Z_y/k_p - D - u_{\text{ref}}$ is obtained. Here, Z_u and Z_y are negative and positive respectively for a positive $D + u_{\text{ref}}$ because $u_{\text{ref}} + D$ decreases the zero-frequency quantity of the process input while it increases that of the process output. Then, $|Z_u(k+1)|$ is smaller than $|u_{\text{ref}}(k) + D|$, as shown in Figure 13.7b. So, the discrete-time overall process has the following transfer function:

$$Z_u(z) = -z^{-1}\beta\hat{u}_{\text{ref}}(z) \quad (13.39)$$

where $\hat{u}_{\text{ref}}(k) = u_{\text{ref}}(k) + D$ and $\hat{u}_{\text{ref}}(z)$ is the z -transform of $\hat{u}_{\text{ref}}(k)$. β is positive and less than unity. The update method with a positive α has the following transfer function:

$$\hat{u}_{\text{ref}}(z) = \frac{\alpha z^{-1}Z_u(z)}{1 - z^{-1}} + \frac{\hat{u}_{\text{ref}}(0)}{1 - z^{-1}} \quad (13.40)$$

Then, the following closed-loop transfer function is obtained from (13.39) and (13.40):

$$Z_u(z) = \frac{-\beta\hat{u}_{\text{ref}}(0)}{1 - z^{-1} + z^{-2}\alpha\beta} \quad (13.41)$$

It is straightforward to derive the condition of $0 \leq \alpha \leq 1/\beta$ for a stable update from (13.41). This means that $\alpha = 1$ will stabilize the closed-loop response because β is less than unity.

Example 13.3

The following process is simulated to confirm the performance of Relay_SS:

$$G(s) = \frac{(zs + 1)\exp(-0.3s)}{(s + 1)^2} \quad (13.42)$$

Table 13.11 shows the estimated amplitude ratio and frequency corresponding to the prespecified phase angle. As expected, the proposed method with $\tau_\alpha = 3$ provides almost exact estimates for the range $-\pi/2 \leq \angle G(i\omega) \leq -\pi$. If a bigger τ_α is chosen and there are no numerical errors, then Relay_SS would provide exact estimates. Table 13.12 shows the MATLAB code for Example 13.3.

Table 13.11 Estimated frequency responses for the SOPTD process of $G_p(s) = (zs + 1)\exp(-0.3s)/(s + 1)^2$. Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, Korean Journal of Chemical Engineering Copyright © [2008] Korean Institute of Chemical Engineer.

$\angle G(i\omega)$	Process ($z = 0.2$)		Relay_SS ($z = 0.2$)		Process ($z = -0.2$)		Relay_SS ($z = -0.2$)	
	$ G(i\omega) $	ω	$ G(i\omega) $	ω	$ G(i\omega) $	ω	$ G(i\omega) $	ω
$-\pi$	0.079	3.884	0.079	3.876	0.225	1.940	0.226	1.936
$-3\pi/4$	0.240	1.849	0.242	1.859	0.419	1.207	0.420	1.205
$-\pi/2$	0.555	0.911	0.558	0.908	0.678	0.700	0.679	0.700

Table 13.12 MATLAB code to simulate Table 13.11.

```

                                relay_SS_ex1.m
clear; tf=30.0; delt=0.01; phase=pi/4;
u=zeros(1,1000); relay1=0.0; x=zeros(2,1);
tf_k=round(tf/delt); rand('seed',0); noise=2*(rand(1,tf_k)-0.5);
yref=0.2; dn=0; dis=0.0; d=1.0;
%phase angle=-pi+phase, dead_zone=dn*delt, relay magnitude=d, dis=
disturbance
alpha=3.0; nmag=0.0; hys=2*nmag+0.01;%uref updata rate,noise magnitude,
hysteresis
index_up=1; index_down=1; index_u_update=1; %no uref update-0, uref
update-1
y=0.0; yb=0.0; t_on=0.0; t_off=0.0; Pon=0.0; Poff=Pon;
amax=0; amin=0; number=-4; t_on_add=0.0; t_off_add=0.0;
off_add=0; on_add=0; sul=0.0; uref=0.0; y_delta=0.1;

for k=1:tf_k
    t=(k-1)*delt; T(k)=t; Y(k)=y; U(k)=(relay1+index_u_update*uref);
    Uref(k)=uref; Yref(k)=yref; for i=1:999 u(i)=u(i+1); end
    if(number>-4 & y-yref>=hys) index_up=1; end;
    if(number>-4 & y-yref<=-hys) index_down=1; end
    if((y-yref<=y_delta) & (number== -4))
        relay1=d; if (y-yref>=y_delta-hys) index_up=1; number=-3; end
    end
    if((y-yref>0.0) & (number== -3) & (index_down==1))
        relay1=-d; number=-2; t_off=t; index_down=0; index_up=0;
    end
    if((y-yref<=0.0) & (number== -2) & (index_up==1))
        relay1=d; number=-1; t_on=t; index_down=0; index_up=0;
    end
    if((y-yref>0.0) & (number== -1) & (index_down==1))
        relay1=-d; number=0; t_off=t; Pon=t-t_on; index=0; index_down=1;
        index_up=0;
    end
end
if(number>=-1)

```

Table 13.12 (Continued)

```

sul=sul+relay1*delt;
if((y-yref)>=0.0 & (yb-yref)<0.0 & (index_down==1))
    t_off_expect=t_on+Pon;
    if (t_off_expect-t>0.0) t_off_add=t_off_expect-t; else t_off_add=0;
end
    t_off=t; Pon=t-t_on; index=0; index_down=0; index_up=0;
end
if ((y-yref)<=0.0 & (yb-yref)>0.0 & (index_up==1))
    index_down=0; index_up=0;
    t_on_expect=t_off+Poff; if (t_on_expect-t>0.0) t_on_add=t_on_
expect-t; else t_on_add=0; end
    t_on=t; Poff=t-t_off; index=1; number=number+1;
    amax_s=amax; amin_s=amin; amax=0.0; amin=0.0; P=Pon+Poff; su=sul;
    if(number>=0)
        w=2*pi/P; kd=1.5*(exp(-(number-1)/3))*su/P;
        if(kd<1.5*d) uref=uref+kd; else uref=uref+1.5*d; end
    end
    sul=0.0;
end
if(amax<(y-yref)) amax=y-yref; end; if(amin>(y-yref)) amin=y-yref;
end
if(number>=1)
    if(index==0)
        w=pi/Poff;
        if(t-t_off<t_off_add-dn*delt) off_add=-4*d*1/pi; else
off_add=0.0; end
        relay1=4*d*sin(w*(t-t_off)-pi-phase)/pi+off_add;
        if(Poff<(t-t_off) & Poff+dn*delt>=(t-t_off) & (y-yref)>0.0)
            relay1=4*d*sin(w*Poff-pi-phase)/pi;
        end
        if(Poff+dn*delt<(t-t_off) & (y-yref)>0.0)
            relay1=4*d*sin(w*(t-t_off)-pi-phase)/pi-4*d/pi;
        end
    end
    if(index==1)
        w=pi/Pon;
        if(t-t_on<t_on_add-dn*delt) on_add=4*d*1/pi; else on_add=0.0;
    end
        relay1=-4*d*sin(w*(t-t_on)-pi-phase)/pi+on_add;
        if(Pon<(t-t_on) & Pon+dn*delt>=(t-t_on) & (y-yref)<0.0)
            relay1=-4*d*sin(w*Pon-pi-phase)/pi;
        end
        if(Pon+dn*delt<(t-t_on) & (y-yref)<0.0)
            relay1=-4*d*sin(w*(t-t_on)-pi-phase)/pi+4*d/pi;
        end
    end
end
end

```

Table 13.12 (Continued)

<pre> if (t>=0) dis=0.0; end u(1000)=relay1+index_u_update*uref+dis; yb=y; [x,y]=g_relay_SS_ex1(x,delt,u); y=y+nmag*noise(1,k); end figure(3); plot(T,U,'g',T,Y,'b',T,Yref,'k',T,Uref,'r'); Period=P; w=2*pi/P; s=j*w; PA=(-pi+phase)*180/pi; Model_AR=(amax_s-amin_s)/(2/(4*d/pi)); Model_w=w; [war]=ar_angle(phase); %Process_PA=(-pi+phase)*180/pi Process_AR=ar; Process_w=w; AR_error=100*(Process_AR-Model_AR)/Process_AR; w_error=100*(Process_w-Model_w)/Process_w; fprintf('Model: w=%5.3f AR=%5.3f PA=%5.4f\n',Model_w,Model_AR,PA); fprintf('Process: w=%5.3f AR=%5.3f PA=%5.4f\n',Process_w,Process_AR, PA); </pre>	
<pre> g_relay_SS_ex1.m function [next_x,y]=g_relay_SS_ex1(x,delt,u); subdelt=delt/10.0; n=round(delt/subdelt); A=[0 -1; 1 -2]; B=[1; 0.2]; C=[0 1]; delay=0.3; delay_k=round(delay/delt); for i=1:n dx=A*x+B*u(1000-delay_k); x=x+dx*subdelt; end next_x=x; y=C*x; return </pre>	<pre> ar_angle.m function [war]=ar_angle(phase) w=0.0; del=0.01; for i=1:100000 w=w+del; s=j*w; [G]=process(s); PA=angle(G); if (imag(G)>0.0) PA=-pi-2*pi+angle(G); end if (PA<(-pi+phase)) break; end end end a=w-del; b=w+del; for i=1:1000 c=(a+b)/2; s=j*a; [G]=process(s); PAa=angle(G)+pi-phase; s=j*c; [G]=process(s); PAc=angle(G)+pi-phase; if (PAa*PAc<0.0) b=c; else a=c; end if (abs(PAa)<0.0000001) break; end end ar=abs(G); w=c; return function [G]=process(s) G=exp(-0.3*s)*(0.2*s+1)/((s+1)^2; return </pre>
<pre> command window >> relay_SS_ex1 Model: w=1.848 AR=0.232 PA=-135.0000 Process: w=1.859 AR=0.239 PA=-135.0000 </pre>	

Table 13.13 Comparisons of Relay_SS with previous methods in estimating the ultimate data corresponding to $\angle G(i\omega) = -\pi$. Enhanced Frequency Response Estimator to Guarantee Pre-specified Phase Angle and Static Disturbance Rejection with All Harmonics Removed, Sung and Lee, *Korean Journal of Chemical Engineering* Copyright ©[2008] Korean Institute of Chemical Engineers.

$G(s) = \frac{\exp(-\theta)}{s+1}$	Process		Conventional		Modified		Relay_SS	
	$ G(i\omega) $	ω	$ G(i\omega) $	ω	$ G(i\omega) $	ω	$ G(i\omega) $	ω
$\theta = 0.1$	0.061	16.32	0.075	16.36	0.066	16.32	0.061	16.31
$\theta = 0.2$	0.118	8.444	0.143	8.537	0.125	8.468	0.118	8.440
$\theta = 1.0$	0.442	2.029	0.486	2.107	0.437	2.040	0.444	2.024
$\theta = 3.0$	0.774	0.819	0.746	0.856	0.792	0.821	0.776	0.818

Example 13.4

The following process is simulated to compare Relay_SS with the previous methods of the conventional relay feedback method and Sung *et al.* (1995):

$$G(s) = \frac{\exp(-\theta s)}{s+1} \tag{13.43}$$

Equation (13.43) is chosen because it usually shows bigger modeling errors than other high-order processes for the harmonics included in the process input. Table 13.13 shows the estimated amplitude ratio and frequency corresponding to $\angle G(i\omega) = -\pi$. As expected, the proposed method provides the best estimates in most cases.

13.2.2 Concluding Remarks

Using Relay_SS to solve the harmonics problem of the previous approaches completely is introduced. It can remove all the harmonics by using the sinusoidal signal directly, of which the frequency is automatically updated to guarantee the desired phase angle. Relay_SS also rejects the effects of static disturbances by adjusting the reference value of the sinusoidal signal.

Problems

- 13.1 Activate the process $G(s) = 1/(s + 1)^5$ using Relay_PS in Section 13.1 and estimate the frequency response using the describing function analysis in Chapter 8.
- 13.2 Activate the virtual process of Process 3 (refer to the Appendix for details) using Relay_PS in Section 13.1 and estimate the frequency response using the describing function analysis in Chapter 8. Tune the PID controller using the ZN tuning rule and demonstrate the control performance.
- 13.3 Activate the following Hammerstein nonlinear process using Relay_PS combined with the disturbance rejection technique in Section 13.1 and estimate the ultimate frequency of the linear dynamic subsystem and the input nonlinear function. Compare the model output and the process output.

$$v(t) = 1 - \exp(-0.3u(t)) + 0.2u(t)$$

$$\frac{d^3y(t)}{dt^3} + 3\frac{d^2y(t)}{dt^2} + 3\frac{dy(t)}{dt} + y(t) = 2.0v(t - 0.2)$$

- 13.4 Activate the process of $G(s) = 1/(s + 1)^5$ using Relay_SS in Section 13.2 with $y_{\text{ref}} = 0.3$ and estimate the frequency response for phase angles of $-\pi$, $-3\pi/4$, $-\pi/2$ using the describing function analysis in Chapter 8.
- 13.5 Activate the virtual process of Process 3 (refer to the Appendix for details) using Relay_SS in Section 13.2 with $y_{\text{ref}} = 0.3$ and phase angle of $-\pi$ and estimate the frequency response using the describing function analysis in Chapter 8. Tune the PID controller using the ZN tuning rule and demonstrate the control performance.

References

- Atherton, D.P. (1975) *Nonlinear Control Engineering*, Van Nostrand Reinhold, London.
- Je, C.H., Lee, J., Sung, S.W. and Lee, D.H. (2009) Enhanced process activation method to remove harmonics and input nonlinearity. *Journal of Process Control*, **19**, 353.
- Lee, T.H., Wang, Q.G. and Tan, K.K. (1995) A modified relay-based technique for improved critical point estimation in process control. *IEEE Transactions on Control Systems Technology*, **3**, 330.
- Park, J.H., Sung, S.W. and Lee, I. (1997) Improved relay auto-tuning with static load disturbance. *Automatica*, **33**, 711.
- Park, H.C., Sung, S.W. and Lee, J. (2006) Modeling of Hammerstein–Wiener processes with special input test signals. *Industrial & Engineering Chemistry Research*, **45**, 1029.
- Shen, S., Yu, H. and Yu, C. (1996) Use of saturation-relay feedback for autotune identification. *Chemical Engineering Science*, **51**, 1187.
- Sung, S.W. and Lee, J. (2006) Relay feedback method under nonlinearity and static disturbance conditions. *Industrial & Engineering Chemistry Research*, **45**, 4028.
- Sung, S.W. and Lee, J. (2008) Enhanced frequency response estimator to guarantee pre-specified phase angle and static disturbance rejection with all harmonics removed. *Korean Journal of Chemical Engineering*, **25**, 1273.
- Sung, S.W., Park, J.H. and Lee, I. (1995) Modified relay feedback method. *Industrial & Engineering Chemistry Research*, **34**, 4133.
- Tan, K.K., Lee, T.H., Huang, S. *et al.* (2006) Improved critical point estimation using a preload relay. *Journal of Process Control*, **16**, 445.

Appendix

Use of Virtual Control System

A virtual control system composed of a user's MATLAB codes and virtual processes is used to emulate real situations in industry. This can contribute significantly to improving the ability to solve real problems in industry. Students can practice the whole procedure of process activation, modeling, controller design and implementation on the basis of a virtual control system. This appendix introduces the virtual control system and several examples of user codes.

A.1 Setup of the Virtual Control System

The overview, installation and setting the parameters of the virtual control system are briefly explained in this section.

A.1.1 Overview

The virtual control system is composed of a buffer, a user's code and a virtual process. Figure A.1 shows a schematic diagram for data exchange between the user's code and the virtual process.

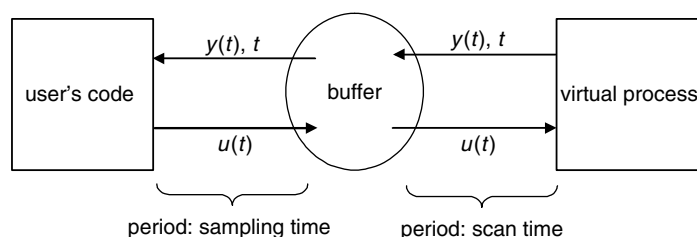


Figure A.1 Schematic diagram of the virtual control system.

The virtual process is coded using Visual Basic and the user's code is a MATLAB m-file. The buffer is one of several forms of database, Excel, clipboard and data file, of which the role is temporal data storage to incorporate the data exchange between the user's code and the virtual process. The user's code periodically sends the control output $u(t)$ to the buffer and gets the time t and the process output $y(t)$ from the buffer with the period of the sampling time. The virtual process periodically sends t and $y(t)$ to the buffer and gets $u(t)$ from the buffer with the period of the scan time. The sampling time should be larger than the scan time. Also, it is recommended to choose the sampling time as a multiple of the scan time.

A.1.2 Installation

Installation of the virtual control system is very simple. Just double click on Setup.exe in the package directory and input the directory name into which all the files are extracted. Let us call the directory "virtual system directory." To run the virtual control system, click on the Windows start menu and choose the virtual control system or double click on Virtual Processes.exe in the virtual system directory. MATLAB functions (get_t, get_y and set_u) for the data exchange between the user's code and the virtual process are provided in the virtual system directory so that the user can exchange their data with the virtual processes easily without knowing the details of the system. The present working directory of MATLAB should be the virtual system directory. Also, the user's code should be placed in the virtual system directory. If the user wants to use another directory, then they should make sure that all the files in the virtual system directory are placed in the directory and double click on the Virtual Processes.exe to run the virtual processes (do not use the Windows start menu in this case).

A.1.3 Setting the Virtual Processes

Figure A.2 shows the virtual processes. The user can choose one of the available virtual processes from the menu in Figure A.2a. Figure A.2b shows the chosen virtual process. If the user clicks on the button again, the virtual process will be reset and restarted.

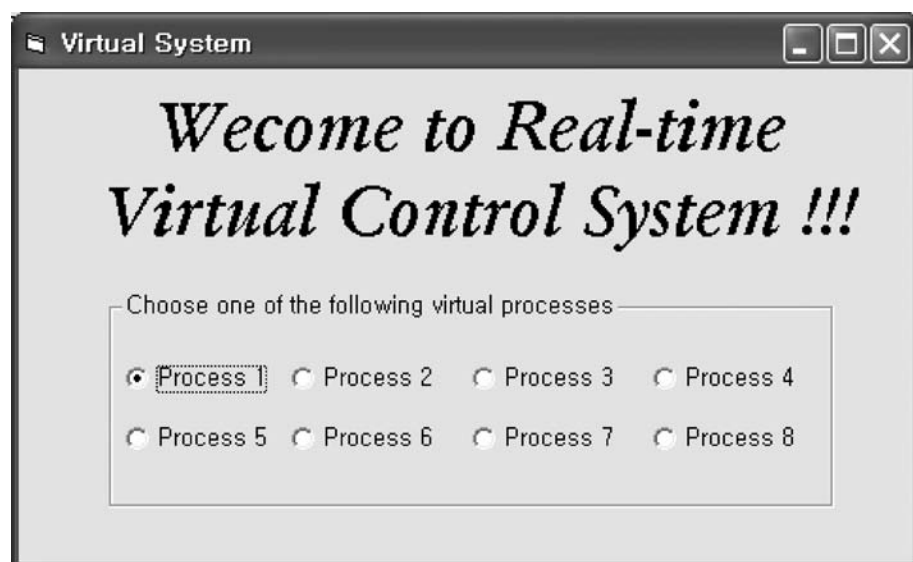
The user can set the parameters of the virtual process, such as the upper limit and lower limit of the process input, the scan time and scaling of the axes by inputting a numeric value to the text box and clicking on the Press to Set button.

A.2 Examples

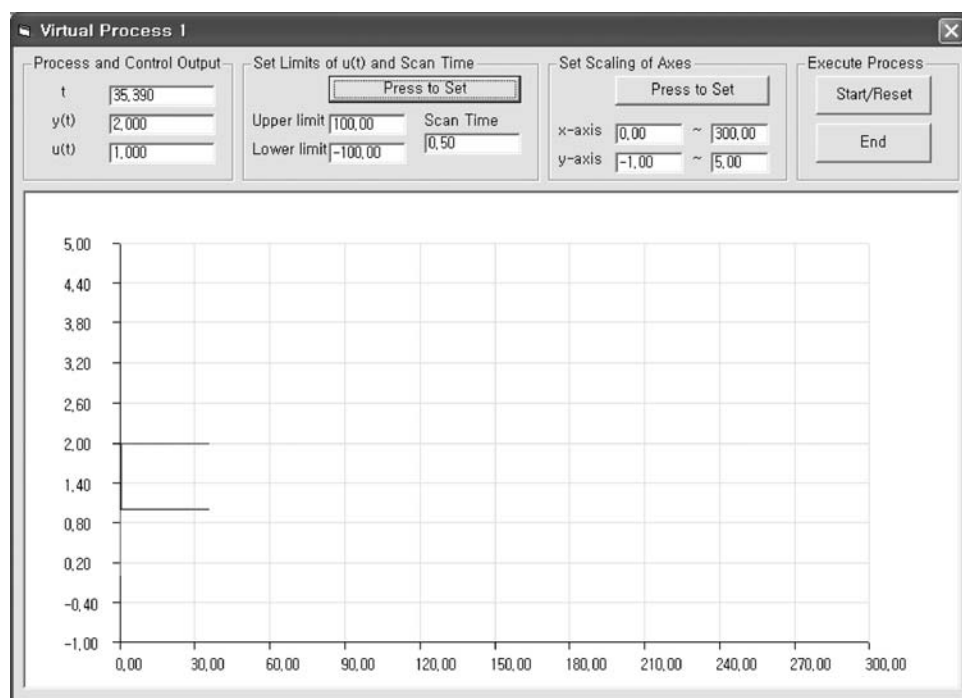
Several examples of user codes are introduced to demonstrate how to use the virtual control system.

A.2.1 Process Activation Using a Step Signal

Table A.1 shows the user's MATLAB code to activate a virtual process using a step signal $u(t) = 3$ for $20 \leq t$ and $u(t) = 1$ for $t < 20$. Figure A.3 shows the dynamic behavior of the virtual control system for the step input test. The process output measured by the user's code in Table A.1 and the process input set by the user's code are shown in Figure A.4.



(a)



(b)

Figure A.2 Virtual control system: (a) menu to select the virtual process; (b) selected virtual process.

Table A.1 User's MATLAB code to activate a virtual process using a step signal.

```
my_step_test.m
function [data_t data_y data_u]=my_step_test(tf)
sampling_time = 0.5; %sampling time
[t]=get_t(); %time reading
n_tf=round(tf/sampling_time); t_ref=t; i=0;
while t<=tf
    [t]=get_t(); % Time reading
    delt=t-t_ref;
    if (delt>=sampling_time*0.99)
        i=i+1; t_ref=t;
        % Beginning of my control algorithm
        [y]=get_y(); %process output reading
        if (20.0<=t) u=3.0; else u=1.0; end % controller output
        set_u(u); % Control output sending
        % End of my control algorithm
        data_t(i)=t; data_u(i)=u; data_y(i)=y; % data storage
    end
end
figure(1); plot(data_t,data_y);
figure(2); plot(data_t,data_u);

command widow
>> [data_t data_y data_u]=my_step_test(100);
```

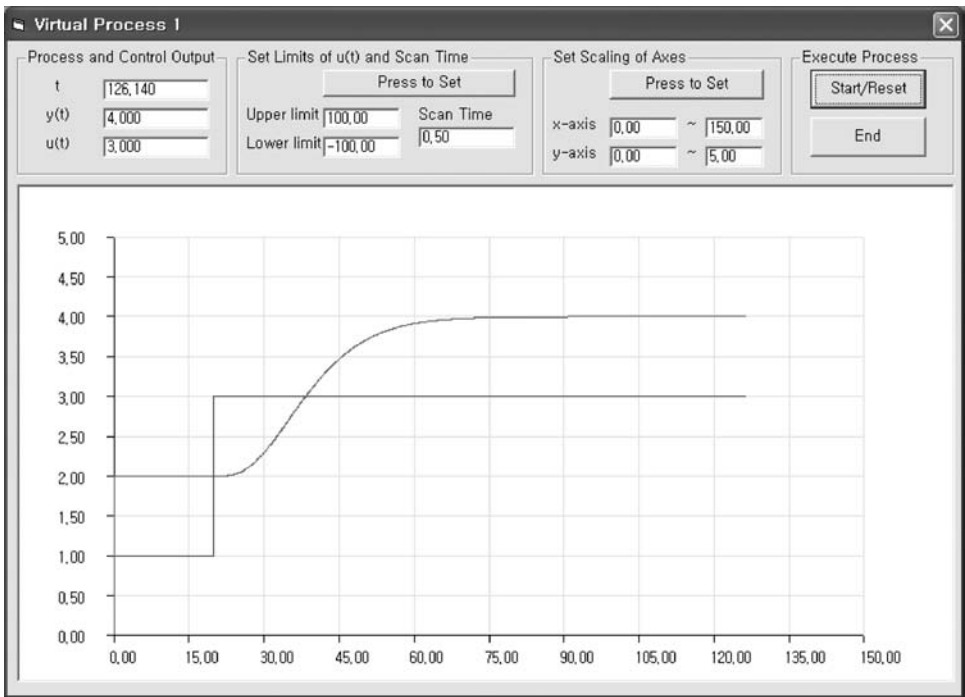


Figure A.3 Response of the virtual process for the step input test.

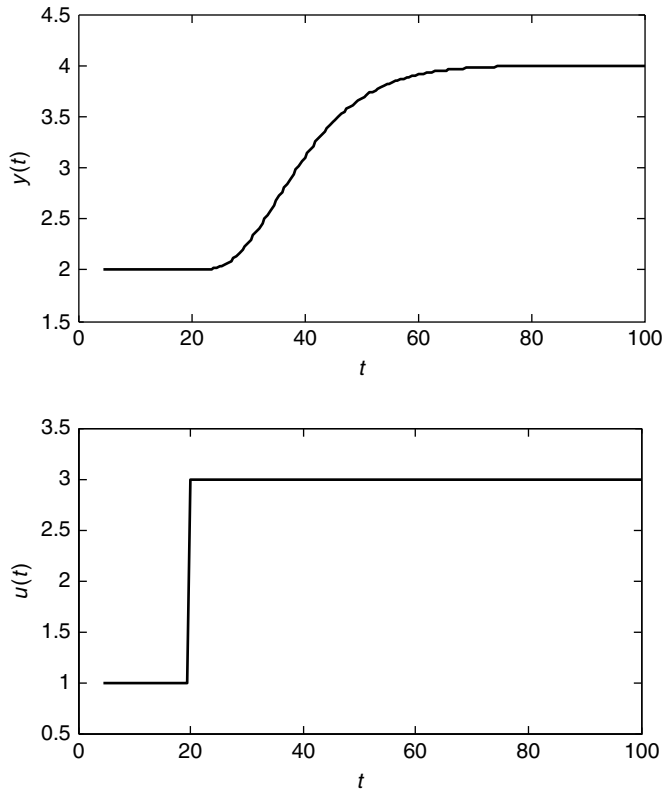


Figure A.4 The process output and the process input obtained by the user's MATLAB code in Table A.1.

Table A.2 User's MATLAB code to control a virtual process using a PID controller.

```

                                my_pid.m
function [data_t data_y data_u]=my_pid(tf)
sampling_time = 0.5;
[t]=get_t(); %Time reading
n_tf=round(tf/sampling_time); t_ref=t; i=0;

[yb]=get_y(); %process output reading
kc=1.2; ti=10.0; td=6.0; s=1.0; ysb=2.0;
while t<=tf
    [t]=get_t(); % Time reading
    delt=t-t_ref;
    if (delt>=sampling_time*0.99)
        i=i+1; t_ref=t;
    end
end

```

(continued)

Table A.2 (Continued)

```
% Beginning of my control algorithm
[y]=get_y(); %process output reading
if(20.0<=t) ys=3.0; else ys=2.0; end % setpoint
s=s+kc*(ys-y)*delt/ti;
u=kc*(ys-y)+s+kc*td*((ys-y)-(ysb-yb))/delt; %control output
ysb=ys; yb=y;
set_u(u); % Control output sending
% End of my control algorithm
data_t(i)=t; data_u(i)=u; data_y(i)=y; % data storage
end
end
figure(1); plot(data_t,data_y);
figure(2); plot(data_t,data_u);

command widow
>> [data_t data_y data_u]=my_pid(150);
```

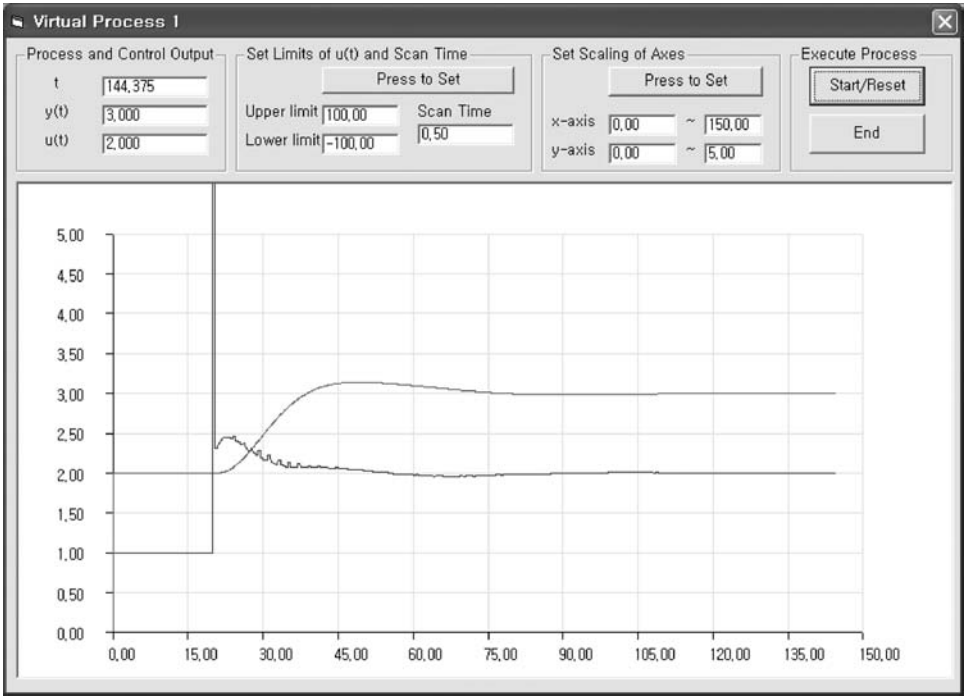


Figure A.5 Response of the virtual process for the PID controller.

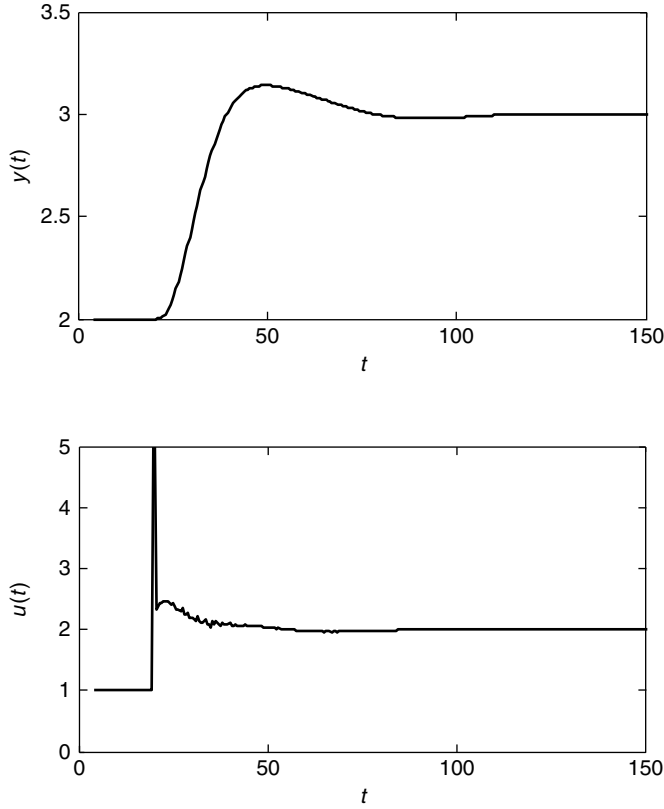


Figure A.6 The process output and the process input obtained by the user’s MATLAB code in Table A.2.

A.2.2 Process Control Using a Proportional–Integral–Derivative Controller

Table A.2 shows the user’s MATLAB code to control a virtual process using a PID controller for which the parameters are $k_c = 1.2$, $\tau_i = 10.0$ and $\tau_d = 6.0$ and the setpoint is 3.0 for $20 \leq t$. Figure A.5 shows the response of the virtual processes for the PID controller. The process output measured by the user’s code in Table A.2 and the process input set by the user’s code are shown in Figure A.6.

A.2.3 Process Activation Using the Relay Feedback Method

Table A.3 shows the user’s MATLAB code to activate a virtual process using the relay feedback method with $u(t) = 3$ for $y(t) < 2.0$ and $u(t) = 0$ for $y(t) \geq 2.0$. The relay feedback starts at $20.0 \leq t$. Figure A.7 shows the response of the virtual processes for the relay feedback method. The process output measured by the user’s code in Table A.3 and the process input set by the user’s code are shown in Figure A.8.

Table A.3 User's MATLAB code to activate a virtual process using the relay feedback method.

```

                                my_relay.m
function [data_t data_y data_u]=my_relay(tf)
sampling_time = 0.5;
[t]=get_t(); %Time reading
n_tf=round(tf/sampling_time); t_ref=t; i=0;

while t<=tf
    [t]=get_t(); % Time reading
    delt=t-t_ref;
    if (delt>=sampling_time*0.99)
        i=i+1; t_ref=t;
        % Beginning of my control algorithm
        [y]=get_y(); %process output reading
        if(t>20.0)
            if (3.0<=y) u=0.0; else u=3.0; end % relay
        else
            u=1.0;
        end
        set_u(u); % Control output sending
        % End of my control algorithm
        data_t(i)=t; data_u(i)=u; data_y(i)=y; % data storage
    end
end
figure(1); plot(data_t,data_y);
figure(2); plot(data_t,data_u);

```

```

                                command widow
>> [data_t data_y data_u]=my_relay(150);

```

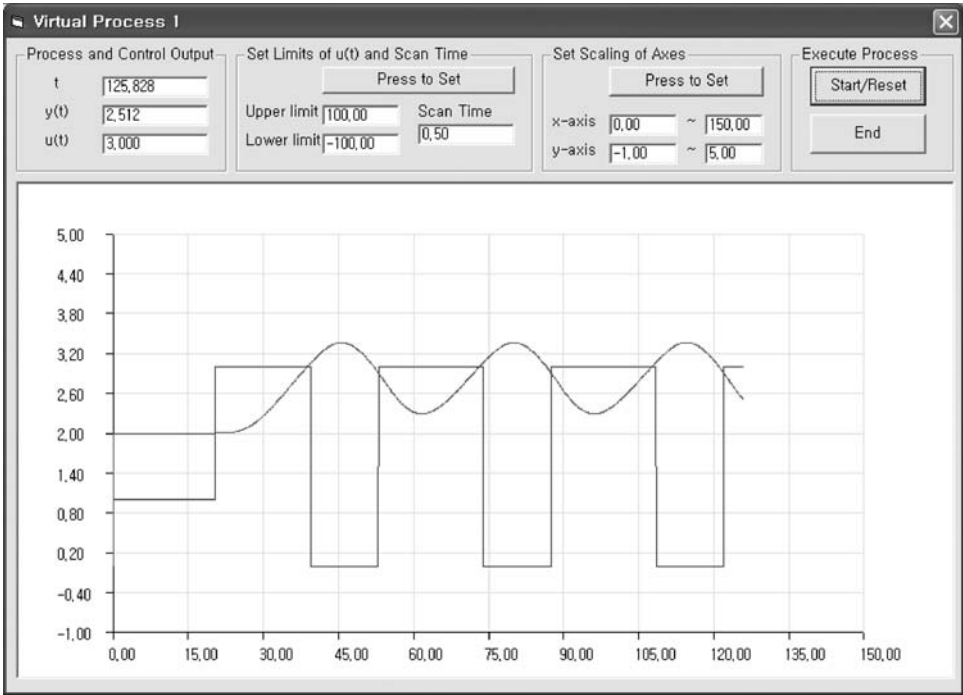


Figure A.7 Response of the virtual process for the relay feedback method.

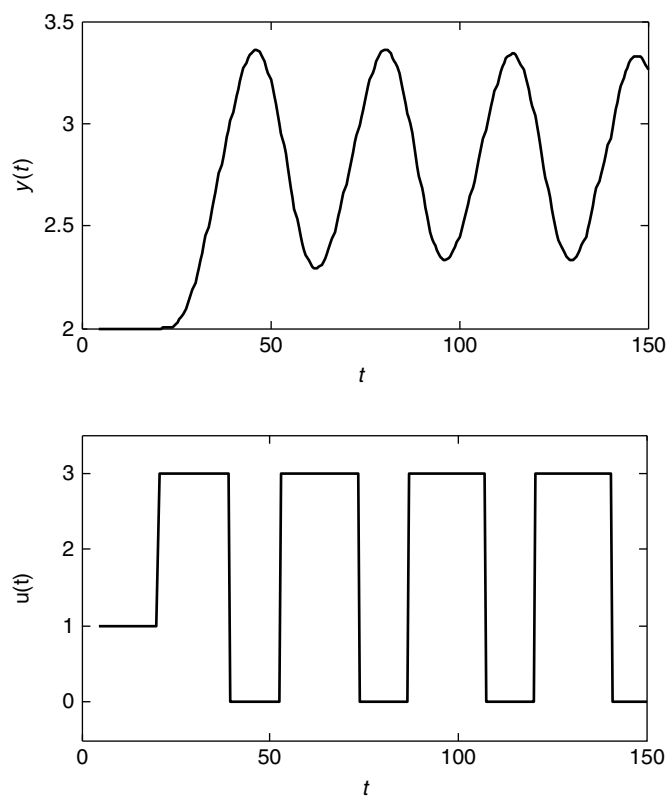


Figure A.8 The process output and the process input obtained by the user’s MATLAB code in Table A.3.

Index

- amplitude ratio, 95, 98, 241, 243, 246, 267, 268, 338
- anti-derivative kick, 141
- anti-windup, 130
 - conditional integration, 131
 - back-calculation, 134
- Autoregressive Exogenous Input (ARX) model, 317, 319
- autotuning, 241

- back-calculation, 134
- biased-relay, 248, 250, 351
- bisection method, 64, 98
- Black-box model, 4–5
- block diagram, 92
- bode plot, 99, 203, 207
- bode stability, 203

- cascade control, 187, 215
 - design (tuning), 187
 - primary controller, 215
 - secondary controller, 215
- characteristic equation, 202, 219, 223
- closed-loop
 - control system, 57, 83, 201
 - transfer function, 201
- commercial PID controllers, 135
- conditional integration, 131
- continuous-cycling method, 154
- continuous-time, 275, 337
- controlled variables, 3
- convolution theorem, 24
- critical frequency, 203
- critically damped process, 82
- cyclic-steady-state, 5, 95, 240, 250, 346

- damping factor, 82
- decay ratio, 83
- decoupled control structure, 220
- derivative kick, 141
- derivative time, 112
- describing function analysis, 241, 377
- deviation variable, 7
- difference equation, 317
- differential equation, 4, 20, 29, 31, 35, 45, 275
- discrete-time, 317, 337
- dynamic behaviors, 79

- equivalent gain plus time delay, 220
- equivalent time delay, 175
- Euler formula, 88, 89, 96
- Euler method, 47, 69, 113

- first order plus time delay (FOPTD), 79, 84, 159, 161, 174, 339
- frequency response, 19, 94, 99, 157, 170, 235, 240, 247, 250, 263, 338, 367
 - amplitude ratio, 95, 98, 241, 243, 246, 267, 268, 338
 - phase angle, 95, 99, 174, 241, 243, 338, 368
- frequency response analysis, 240, 261
- Fourier analysis, 247
- Fourier series, 235

- gain crossover frequency, 203, 210
- gain margin, 210
- gain scheduling, 225

- Hammerstein process, 382
- IMC tuning rule, 159
- implementation, 113
- impulse function, 21
- integral time, 112
- integral transform, 275
- integral windup, 129
- integrating process, 91, 126, 197, 227
- interacting PID controller, 137
- internal feedback loop, 188, 215, 227
- interval having method, 70, 321, 328
- inverse Laplace transform, 16, 26
- inverse response, 91
- ITAE-1 tuning rule, 161
- ITAE-2 tuning rule, 167
- laplace transform, 16
- least squares method, 59, 173, 176, 277, 281, 320, 338, 358
- left-half-plane (LHP) pole, 90
- Levenberg-Marquardt method, 72, 297, 326
- linear process, 9
 - time-invariant, 9
 - time-variant, 9
- linearization, 13
- low order plus time delay process, 79
 - first order plus time delay (FOPTD), 79, 84, 159, 161, 174, 339
 - second order plus time delay (SOPTD), 82, 166, 169, 171, 175, 231
- manipulating variables, 3–4
- margin
 - phase, 210
 - gain, 210
- marginally stable, 154, 203–204, 243
- model conversion, 337
- modeling error, 219
- model reduction, 170, 338
- modified Fourier transform, 250
- Newton-Raphson method, 65
- noise suppressing PID controller, 141
- non-interacting PID controller, 135
- numerical
 - analysis, 59
 - derivative, 45, 59, 68, 113, 301, 326
 - integration, 68, 112, 247, 253, 276
- Nyquist plot, 99, 207
- Nyquist stability, 207
- offset, 124, 153, 219, 229
- open-loop stable process, 91, 97, 122, 153, 230
- open-loop transfer function, 201, 203, 207
- open-loop unstable process, 91
- optimal gain margin tuning rule, 169
- optimization method, 69
 - interval having method, 70, 321, 328
 - Levenberg-Marquardt method, 72, 297, 326
- output error (OE) model, 318, 325
- over-damped process, 82
- overshoot, 83
- Padé approximation, 25
- parallel PID controllers, 138
- partial fraction, 26
- periodic function, 25, 235, 248, 254
- phase angle, 95, 99, 174, 241, 243, 338, 368
- phase crossover frequency, 204
- phase margin, 210
- PID controllers, 111
 - commercial PID, 135
 - derivative time, 112
 - implementation, 113
 - integral time, 112
 - interacting PID, 137
 - noise suppressing PID, 141
 - non-interacting PID, 135
 - parallel PID, 138
 - proportional band, 112
 - setpoint, 112
 - two-degree-of-freedom PID, 140
 - unified structure of PID, 145
- PID controller tuning, 151
 - IMC tuning rule, 159
 - ITAE-1 tuning rule, 161
 - ITAE-2 tuning rule, 167
 - modeling error, 196
 - model reduction, 170
 - optimal gain margin tuning rule, 169
 - trial and error tuning, 151
 - Ziegler-Nichols tuning, 157
- poles, 86
- prediction error identification method, 291, 319, 325
- prediction model
 - ARX model, 317
 - OE model, 318

- primary controller, 215
- process activation, 343, 373, 387, 400
- process control, 3
 - controlled variables, 3–4
 - manipulating variables, 3–4
 - process input, 3–4
 - process output, 3–4
- process identification, 4
 - black-box model, 4–5
 - methods, 154, 235, 275, 317
- process input, 3
- process output, 3
- process reaction curve method, 84, 157
- proportional band, 112
- proportional gain, 112
- pulse signals, 373
- relay feedback, 157, 240, 243, 248, 250, 263, 287, 345, 351, 373, 405
 - biased, 248, 250, 287, 351
 - large range of operation, 365
 - modifications, 373
 - nonlinearity and disturbance, 357
 - static disturbance, 352
 - two-channel, 244
 - unbiased, 345
- right-half-plane (RHP) pole, 90, 207
- root-finding method, 63
 - bisection, 63
 - Newton-Raphson, 65
- saturation, 129
- secondary controller, 215
- second order plus time delay (SOPTD), 82, 166, 169, 171, 175, 231, 339
- setpoint, 112
- settling time, 83
- simulations, 45
- sine signal, 387
- Smith predictor, 217
- stable poles, 90
- state space, 32, 48, 114, 292
- static gain, 79, 82, 97, 153
- steady-state, 5, 7, 253, 275
- step function, 21
- step response, 81, 82, 85, 91
- superposition rule, 11, 34, 94, 97, 202
- Taylor series approximation, 13, 14, 25, 175
- time constant, 79, 82
- time delay, 10, 21, 25, 50, 79, 82, 85, 217, 245, 320, 325, 327
- time delay compensator, 217
- time-invariant linear process, 9
- time-variant linear process, 9
- transfer function, 33, 86, 92, 95, 113, 201, 241, 337
- trial and error tuning, 151
- two-degree-of-freedom PID controller, 140
- two-channel-relay, 244
- ultimate
 - amplitude ratio, 97, 156
 - frequency, 98, 154, 157, 174, 176, 241, 244, 264, 339, 347, 377
 - gain, 98, 154, 157, 188, 241, 244, 264, 377
- unbiased-relay, 345
- under-damped process, 83
- unified structure of PID controller, 145
- uniformly distributed random noise, 120, 287, 304, 358, 360
- unstable poles, 90
- unstable process, 91, 169, 175, 188, 197, 227
- unsteady state, 278
- virtual process, 104, 149, 198, 272, 273, 316, 336, 342, 371, 397, 398, 399
- weight, 281, 283
- Weiner process, 360
- zeroes, 86, 91, 175
- Ziegler-Nichols tuning, 157
- z-transform, 337, 393